IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT EXAMINING OPERATION

ATTN'Y DOCKET NO.:	ETS-TCA
APPLICATION OF:	PETER BRITTINGHAM, MARY E. MORLEY, MARK K. SINGLEY, MARK G. ZELMAN, KRISHNA N. JHA, JAMES H. FIFE, ROBERT L. RARICH, IRVIN R. KATZ, RANDY E. BENNETT
FOR:	COMPUTER-BASED TEST-ITEM GENERATION AND CLONING

VISUAL BASIC SOURCE CODE APPENDIX (VBSCA 1-469)

VISUAL BASIC SOURCE CODE APPENDIX
TABLE OF CONTENTS[1]

[1] All software COPYRIGHT 1999 ETS except for MTAPI.BAS

```
' TCA.vbp
Type=Exe
Reference=*\G{00020430-0000-0000-C000-000000000046}#2.0#0#..\..\..\..\WINNT\System32\
StdOle2.Tlb#OLE Automation
Reference=*\G{00020905-0000-0000-C000-000000000046}#8.0#409#..\..\..\Microsoft
Office\Office\MSWORD8.OLB#Microsoft Word 8.0 Object Library
Reference=*\G{953298D7-F0DE-11D2-AED3-000000000000}#13.0#0#AXProlog.exe#AXProl
og
Object={FE0065C0-1B7B-11CF-9D53-00AA003C9CB6}#1.1#0; COMCT232.OCX
Object={6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0; COMCTL32.OCX
Object={BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0; TABCTL32.OCX
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0; COMDLG32.OCX
Form=TCA.frm
Module=Util; modUtil.bas
Class=Model; Model.cls
Class=Constraint; Constraint.cls
Class=Variable; Variable.cls
Class=TCAApplication; Application.cls
Module=StartUp; Main.bas
Form=Variable.frm
Class=CVariables; CVariables.cls
Class=CConstraints; CConstraints.cls
Form=Constraint.frm
Class=MSWord; Word.cls
Form=frmSplash.frm
Class=VarInteger; VarInteger.cls
Class=VarReal; VarReal.cls
Class=VarFraction; VarFraction.cls
Class=VarString; VarString.cls
Form=frmIndexedString.frm
Class=File; File.cls
Class=CClones; CClones.cls
Class=IniFile; IniFile.cls
Class=Win32API; Win32API.cls
Class=CModels; CModels.cls
Class=Clone; Clone.cls
Form=frmAttributes.frm
Class=Family; Family.cls
Class=DocStatus; DocStatus.cls
Class=Checksum; Checksum.cls
Form=frmProgress.frm
Class=Progress; Progress.cls
Form=frmDifficulty.frm
Class=DifficultyEstimate; DifficultyEstimate.cls
```

```
Class=GREDifficultyEstimate; GREDifficultyEstimate.cls
Class=SMCModel; PSModel.cls
Class=QCModel; qcmodel.cls
Class=DSModel; dsmodel.cls
Class=VarUntyped; VarUntyped.cls
Class=LockedItem; LockedItem.cls
Class=GMATDifficultyEstimate; GMATDifficultyEstimate.cls
Form=frmAbout.frm
Form=frmNew.frm
Form=String.frm
Class=SubString; SubString.cls
Class=ConstraintSolver; ConstraintSolver.cls
Class=StringSolver; StringSolver.cls
Class=Value; Value.cls
Class=PrintModel; PrintModel.cls
Module=MTAPI; MTAPI.bas
Module=MTDeclarations; MTDeclarations.bas
Module=MTUtil; MTUtil.bas
Form=frmProlog.frm
ResFile32="Tca.res"
IconForm="frmTCA"
Startup="Sub Main"
HelpFile=""
Title="TCA"
ExeName32="TCA.exe"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0"
MajorVer=0
MinorVer=1
RevisionVer=145
AutoIncrementVer=1
ServerSupportFiles=0
VersionCompanyName="ETS"
CompilationType=0
OptimizationType=2
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
```

```
StartMode=0
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1
```

```
' AXProlog.vbp
Type=OleExe
Reference=*\G{00020430-0000-0000-C000-000000000046}#2.0#0#..\..\..\..\WINNT\System32\
STDOLE2.TLB#OLE Automation
Reference=*\G{3D5C6BF0-69A3-11D0-B393-00A0C9055D8E}#1.0#0#..\..\..\Common
Files\designer\MSDERUN.DLL#Microsoft Data Environment Instance 1.0
Reference=*\G{00000200-0000-0010-8000-00AA006D2EA4}#2.0#0#..\..\..\Common
Files\system\ado\msado20.tlb#Microsoft ActiveX Data Objects 2.0 Library
Class=Prolog; Prolog.cls
Module=Module1; Timer.bas
Class=File; File.cls
Startup="(None)"
HelpFile=""
ExeName32="AXProlog.exe"
Command32=""
Name="AXProlog"
HelpContextID="0"
CompatibleMode="1"
CompatibleEXE32="AXProlog.exe"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="ETS"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=1
Unattended=-1
Retained=0
ThreadPerObject=-1
MaxNumberOfThreads=1
DebugStartupOption=0
```

```
' Common.bas
Attribute VB_Name = "Common"
```

```
' Main.bas
Attribute VB_Name = "StartUp"

Option Explicit

Public Const READ_UNTIL_EOF = 0
Public Const INI_DIRECTORY = "C:\TCS\TCA\OUT\TCAOUT.INI"
Public Const IN_DIRECTORY = "C:\TCS\TCA\IN\"
Public Const OUT_DIRECTORY = "C:\TCS\TCA\OUT\"
Public Const LOCKED_ITEM_NAME = "TCATEMP.DOC"
Public Const LVM_FIRST = &H1000
Public Const LVM_SETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 54
Public Const LVM_GETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 55
Public Const LVS_EX_FULLROWSELECT = &H20
Public Const HALT_FN = "C:\HALT.TCA"

Public Const STRING_DELIMITER = 164

Private Sub Main()

  Dim MyApp As New TCAApplication

  If App.PrevInstance Then
    Call MsgBox("Only one instance of TCA may be run at a time!", _
      vbExclamation, "Error")
    Exit Sub
  End If

  ' 10 seconds for component timeout
  App.OleRequestPendingTimeout = 10000

  MyApp.Run

End Sub
```

```
' modUtil.bas
Attribute VB_Name = "Util"
Option Explicit

' Capitalizes the first letter of a string if it's a lower case letter
Sub CapitalizeString(strInput As String)

   Dim str1, str2 As String
   Dim intStrLen As Integer

   intStrLen = Len(strInput)

   If (intStrLen > 0) Then
      str1 = UCase(left(strInput, 1))
   End If

   If (intStrLen > 1) Then
      str2 = right(strInput, intStrLen - 1)
   End If

   strInput = str1 & str2

End Sub

' Selects contents of text box for easy editing
Sub txtSelectAll(txtTextBox As TextBox)

   ' Automatically select all text
   txtTextBox.SelStart = 0
   txtTextBox.SelLength = Len(txtTextBox.Text)

End Sub

' Checks to see if a file exists

Function FileExists(ByVal strFN As String) As Boolean

   Dim intFNum As Integer

   ' Get the file number
   intFNum = FreeFile

   ' Open the file and trap any errors
   On Error GoTo NotFound
```

```
    Open strFN For Binary Access Read As #intFNum
    On Error GoTo 0

    Close #intFNum

    FileExists = True
    Exit Function

NotFound:

    ' Close the file
    Close #intFNum
    FileExists = False
    Exit Function

End Function

' extracts the path from a path/filename string

Function ExtractPath(ByVal strFN As String) As String

    Dim varI1 As Variant
    Dim varI2 As Variant

    ' find the last "\" in the string
    varI1 = 0
    Do
        varI2 = varI1
        varI1 = InStr(varI2 + 1, strFN, "\")
    Loop Until varI1 = 0

    ExtractPath = Mid(strFN, 1, varI2)

End Function

' extracts the file name from a path/filename string

Function ExtractFileName(ByVal strFN As String) As String

    Dim varI1 As Variant
    Dim varI2 As Variant

    ' find the last "\" in the string
    varI1 = 0
    Do
```

```
        varI2 = varI1
        varI1 = InStr(varI2 + 1, strFN, "\")
    Loop Until varI1 = 0

    ExtractFileName = Mid(strFN, varI2 + 1, Len(strFN) - varI2)

End Function

' extracts the file name sans extension from a path/filename string
Function ExtractFileNameNoExt(ByVal strFN As String) As String

    strFN = ExtractFileName(strFN)

    Dim varI1 As Variant
    Dim varI2 As Variant

    ' find the last "." in the string
    varI1 = 0
    Do
        varI2 = varI1
        varI1 = InStr(varI2 + 1, strFN, ".")
    Loop Until varI1 = 0

    ExtractFileNameNoExt = Mid(strFN, 1, varI2 - 1)

End Function

' extracts the family name - everything up to $R
Function ExtractFamilyName(ByVal strFN As String) As String

    strFN = ExtractFileName(strFN)

    Dim varI As Variant

    ' find "$R" in the string
    varI = InStr(1, strFN, "$R")

    If varI > 0 Then
        ExtractFamilyName = Mid(strFN, 1, varI - 1)
    End If

End Function

' extracts the key, meaning $R and everthing up to the .
Function ExtractFamilyKey(ByVal strFN As String) As String
```

```
    strFN = ExtractFileName(strFN)

    Dim varI As Variant
    Dim varI1 As Variant
    Dim varI2 As Variant

    ' find  "$R" in the string
    varI = InStr(1, strFN, "$R")

    ' find the last "." in the string
    varI1 = 0
    Do
      varI2 = varI1
      varI1 = InStr(varI2 + 1, strFN, ".")
    Loop Until varI1 = 0

    ExtractFamilyKey = Mid(strFN, varI, varI2 - varI)

End Function

' trim nulls off the end of a string
Function TrimAtFirstNull(ByVal strS As String) As String

    Dim varI As Variant

    varI = InStr(1, strS, Chr(0))
    TrimAtFirstNull = left(strS, varI - 1)

End Function

' returns a string with all instances of strFrom replaced
' with strTo in string strS
Function ReplaceAll(ByVal strS As String, ByVal strFrom As String, _
    ByVal strTo As String) As String

    Dim varI As Variant
    Dim intL As Integer

    Do
      varI = InStr(1, strS, strFrom)
      If varI > 0 Then ' found strFrom
        intL = Len(strS)
        strS = left(strS, varI - 1) & strTo & _
          right(strS, intL - Len(strFrom) - varI + 1)
      End If
    Loop Until varI = 0

    ReplaceAll = strS

End Function

' returns the name of indexed string variables
Function GetIndexedName(ByVal strName As String, _
    ByVal intI As Integer) As String

    GetIndexedName = strName & "." & Trim(Str(intI))

End Function

' Prolog shuts down when this file is created
Sub CreateKillFile()

    Open HALT_FN For Output As #10
    Print #10, "Halt!"
    Close #10

End Sub

' Delete the kill file
Sub DestroyKillFile()

    On Error Resume Next ' if it's not there, Kill will produce an error
    Kill HALT_FN
    err.Clear

End Sub
```

```
' MTAPI.BAS
Attribute VB_Name = "MTAPI"
'mtapi.bas 4.0
'==========================================================================
' (c) Copyright 1992-1999 by Design Science, Inc. All rights reserved
' with the exception that registered MathType owners may alter these
' macros for use by themselves and other registered MathType owners
' provided that:
'   1) The alterations are summarized in a comment directly below this
'      copyright notice. The comment should start with the words
'      "Modified by" and include the name of the person altering the
'      macros, the date of alteration, and that person's email address
'      (if available).
'   2) Persons altering the macros notify Design Science of the nature
'      of any changes they have made.
' These provisions may help us help other customers, and will help us
' continue to provide quality products for you in the future.
'==========================================================================

' version # of this API
Public Const MTAPI_VERSION = 4

' maximum length of file paths, names, etc.
Public Const MTAPI_MAX_PATH = 260

' Picture specifier
Public Type MTAPI_PICT
  mm    As Long
  xExt  As Long
  yExt  As Long
  hMF   As Long
End Type

Public Type RECT
  left     As Long
  top      As Long
  right    As Long
  bottom   As Long
End Type

' Picture dimensions
Public Type MTAPI_DIMS
  baseline As Integer  ' dist of baseline from bottom (points)
  bounds   As RECT     ' bounding rectangle (points)
```

```
End Type

' return codes from MT DLL API

' success, no error
Public Const mtOK = 0
' equation OLE 1.0 object on clipboard
Public Const mtOLE_EQUATION = 1
' Windows metafile equation graphic (not OLE object) on clipboard
Public Const mtWMF_EQUATION = 2
' Macintosh PICT equation graphic (not OLE object) on clipboard
Public Const mtMAC_PICT_EQUATION = 4
' equation OLE 2.0 object on clipboard
Public Const mtOLE2_EQUATION = 8

' error return codes

' can't find MathType application
Public Const mtMT_NOT_FOUND = -1
' can't run the MathType application
Public Const mtMT_CANT_RUN = -2
' the MathType application is the wrong version
Public Const mtMT_BAD_VERSION = -3
' the MathType application is already in use
Public Const mtMT_IN_USE = -4
' the MathType application is not running (i.e. unexpectedly aborted)
Public Const mtMT_NOT_RUNNING = -5
' time ran out waiting for the MathType application to start up
Public Const mtRUN_TIMEOUT = -6
' not equation on clipboard
Public Const mtNOT_EQUATION = -7
' file does not exist or bad pathname
Public Const mtFILE_NOT_FOUND = -8
'  insufficient memory
Public Const mtMEMORY = -9
' bad file
Public Const mtBAD_FILE = -10
' requested data does not exist
Public Const mtDATA_NOT_FOUND = -11
' too many server session open
Public Const mtTOO_MANY_SESSIONS = -12
' could not perform one or more subs
Public Const mtSUBSTITUTION_ERROR = -13
' could not perform translation
Public Const mtTRANSLATOR_ERROR = -14
```

```
' could not set preferences, or invalid preference string
Public Const mtPREFERENCE_ERROR = -15
' other error
Public Const mtERROR = -9999

' options values for MTInitAPI
Public Const mtinitLAUNCH_AS_NEEDED = 0
Public Const mtinitLAUNCH_NOW = 1

' options values for MTGetTranslatorsInfo
Public Const mttrnCOUNT = 1
Public Const mttrnMAX_NAME = 2
Public Const mttrnMAX_DESC = 3
Public Const mttrnMAX_FILE = 4
Public Const mttrnOPTIONS = 5

' options values for MTXFormAddVarSub
Public Const mtxfmSUBST_ALL = 0
Public Const mtxfmSUBST_ONE = 1

' find/replace types for MTXFormAddVarSub substitutions
Public Const mtxfmVAR_SUB_BAD = -1
Public Const mtxfmVAR_SUB_PLAIN_TEXT = 0
Public Const mtxfmVAR_SUB_MTEF_TEXT = 1
Public Const mtxfmVAR_SUB_MTEF_BINARY = 2
Public Const mtxfmVAR_SUB_DELETE = 3

' replace style for MTXFormAddVarSub substitutions when replaceType =
mtxfmVAR_SUB_PLAIN_TEXT
Public Const mtxfmSTYLE_TEXT = 1
Public Const mtxfmSTYLE_FUNCTION = 2
Public Const mtxfmSTYLE_VARIABLE = 3
Public Const mtxfmSTYLE_LCGREEK = 4
Public Const mtxfmSTYLE_UCGREEK = 5
Public Const mtxfmSTYLE_SYMBOL = 6
Public Const mtxfmSTYLE_VECTOR = 7
Public Const mtxfmSTYLE_NUMBER = 8

' options values for MTXFormSetPrefs
Public Const mtxfmPREF_EXISTING = 1
Public Const mtxfmPREF_MTDEFAULT = 2
Public Const mtxfmPREF_USER = 3
Public Const mtxfmPREF_LAST = 3

' options values for MTXFormSetTranslator
```

```
Public Const mtxfmTRANSL_INC_NONE = 0
Public Const mtxfmTRANSL_INC_NAME = 1
Public Const mtxfmTRANSL_INC_DATA = 2
Public Const mtxfmTRANSL_INC_MTDEFAULT = 4

' return values from MTXFormGetStatus
Public Const mtxfmSTAT_PREF = -3
Public Const mtxfmSTAT_TRANSL = -2
Public Const mtxfmSTAT_ACTUAL_LEN = -1

' data sources/destinations for MTXFormEqn
Public Const mtxfmPREVIOUS = -1
Public Const mtxfmCLIPBOARD = -2
Public Const mtxfmLOCAL = -3

' data formats for MTXFormEqn
Public Const mtxfmMTEF = 4
Public Const mtxfmHMTEF = 5
Public Const mtxfmPICT = 6
Public Const mtxfmTEXT = 7
Public Const mtxfmHTEXT = 8

' option values for MTSetMTPrefs
Public Const mtprfMODE_NEXT_EQN = 1
Public Const mtprfMODE_MTDEFAULT = 2
Public Const mtprfMODE_INLINE = 4

' MT API functions
Public Declare Function MTAPIVersion Lib "mt4" (ByVal api As Integer) As Long
Public Declare Function MTInitAPI Lib "mt4" (ByVal options As Integer, ByVal timeout As
Integer) As Long
Public Declare Function MTTermAPI Lib "mt4" () As Long
Public Declare Function MTClearClipboard Lib "mt4" () As Long
Public Declare Function MTEquationOnClipboard Lib "mt4" () As Long
Public Declare Function MTXFormReset Lib "mt4" () As Long
Public Declare Function MTXFormAddVarSub Lib "mt4" ( _
  ByVal options As Integer, _
  ByVal findType As Integer, ByVal find As String, ByVal findLen As Long, _
  ByVal replaceType As Integer, ByVal replace As String, ByVal replaceLen As Long, _
  ByVal replaceStyle As Integer _
) As Long
Public Declare Function MTXFormSetTranslator Lib "mt4" (ByVal options As Integer, _
  ByVal transName As String) As Long
Public Declare Function MTXFormSetPrefs Lib "mt4" (ByVal prefType As Integer, ByVal
prefStr As String) As Long
```

```
Public Declare Function MTSetMTPrefs Lib "mt4" (ByVal mode As Integer, ByVal prefs As
String, _
  ByVal timeout As Integer) As Long
Public Declare Function MTXFormEqn Lib "mt4" ( _
  ByVal src As Integer, ByVal srcFmt As Integer, ByVal srcData As String, ByVal srcDataLen
As Long, _
  ByVal dst As Integer, ByVal dstFmt As Integer, ByVal dstData As String, ByVal dstDataLen
As Long, _
  ByRef dims As MTAPI_DIMS) As Long
Public Declare Function MTXFormGetStatus Lib "mt4" (ByVal index As Integer) As Long
```

```
' MTDeclaration.bas
Attribute VB_Name = "MTDeclarations"
'=============================================================
'Windows API declarations
'=============================================================
Public Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal hwnd As Long,
ByVal lpHelpFile As String, ByVal wCommand As Long, ByVal dwData As Long) As Long
Public Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal
lpLibFileName As String) As Long
Public Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long) As Long
Public Declare Function LoadString Lib "user32" Alias "LoadStringA" (ByVal hInstance As
Long, ByVal wID As Long, ByVal lpBuffer As String, ByVal nBufferMax As Long) As Long
Public Declare Function GetLocaleInfo Lib "kernel32" Alias "GetLocaleInfoA" (ByVal Locale
As Long, ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As
Long
Public Declare Function GetEnvironmentVariable Lib "kernel32" Alias
"GetEnvironmentVariableA" (ByVal lpName As String, ByVal lpBuffer As String, ByVal nSize
As Long) As Long
Public Declare Function SetEnvironmentVariable Lib "kernel32" Alias
"SetEnvironmentVariableA" (ByVal lpName As String, ByVal lpValue As String) As Long
Public Declare Function GetTickCount Lib "kernel32" () As Long

'=============================================================
' Constants for use in Windows API calls
'=============================================================
'----------- Used by GetLocaleInfo ---------------------------
' values for LCType (locale info requested) - used in MTLib.InitLocaleStrs
Public Const Locale_SLanguage As Long = &H2
Public Const Locale_SEngLanguage As Long = &H1001

'=============================================================
' Constants for use in Help calls
'=============================================================
Public Const hlpMSWDPreferences_Dialog = 117
Public Const hlpMSWDEquation_Number_Format_Dialog = 6300
Public Const hlpMSWDFormat_Equations_Dialog = 6500
Public Const hlpMSWDInsert_Equation_Section_Dialog = 114
Public Const hlpMSWDFormat_Equation_Section_Dialog = 116
Public Const hlpMSWDSet_Equation_Preferences_Dialog = 37
Public Const hlpMSWDConvert_Equations_Dialog = 44
Public Const hlpMSWDInsert_Equation_Number_Dialog = 118
Public Const hlpMSWDInsert_Requation_Ref_Dialog = 119

Public Const hlpMSWDWT_SetEqnPrefs = 122
Public Const hlpMSWDWT_InlineEqn = 123
Public Const hlpMSWDWT_CenteredEqn = 124
Public Const hlpMSWDWT_CenteredNumberedEqn = 125
Public Const hlpMSWDWT_EqnNumber = 126
Public Const hlpMSWDWT_EqnRef = 127
Public Const hlpMSWDWT_EqnSec = 128
Public Const hlpMSWDWT_ModEqnSec = 129
Public Const hlpMSWDWT_FormatEqnNum = 130
Public Const hlpMSWDWT_ConvertEqn = 131
Public Const hlpMSWDWT_FormatEqn = 132
Public Const hlpMSWDWT_UpdateEqn = 133
'=====================================================================
' Constants for use in the MathType Commands
'=====================================================================
'---------- Numbers we compare against with MTAPIvers ----------
Public Const mtversMajVerHi = 1279  '0x04ff
Public Const mtversMajVerLo = 1024  '0x0400
Public Const mtversMinVer = 1024    '0x0400

'---------- Registry location codes --------------------------
Public Const mtreg_MT_LANG_LOCATION As String =
"HKEY_CURRENT_USER\Software\Design Science\DSMT4\Config"     'Registry entry for
MathType's curent language
Public Const mtreg_MT_LANG_KEY As String = "AppLang"     'registry key for MathType's
curent language
Public Const mtreg_MT_PROGDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories"    'Registry
entry for MathType's directory
Public Const mtreg_MT_PROGDIR_KEY As String = "ProgDir"    'registry key for MathType's
directory
Public Const mtreg_MT_LANGUAGEDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories"    'Registry
entry for MathType's language support files directory
Public Const mtreg_MT_LANGUAGEDIR_KEY As String = "LastLangDir"    'registry key for
MathType's language support files directory
Public Const mtreg_MT_HELPDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories"    'Registry
entry for MathType's help file directory
Public Const mtreg_MT_HELPDIR_KEY As String = "LastHelpDir"    'registry key for
MathType's help file directory
Public Const mtreg_MT_HELPFILE_LOCATION As String =
"HKEY_CURRENT_USER\Software\Design Science\DSMT4\Config"    'Registry entry for
MathType's help file name
Public Const mtreg_MT_HELPFILE_KEY As String = "HelpFile"    'registry key for
MathType's help file name
```

```
Public Const mtreg_MT_SYSTEMDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories"    'Registry
entry for MathType's system directory
Public Const mtreg_MT_SYSTEMDIR_KEY As String = "LastAppSystemDir"    'registry key
for MathType's system directory
Public Const mtreg_MT_PREFDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories"    'Registry
entry for MathType's preferences folder
Public Const mtreg_MT_PREFDIR_KEY As String = "LastPrefsDir"    'registry key for
MathType's system directory
Public Const mtreg_MT_WORDCMDS_LOCATION As String =
"HKEY_CURRENT_USER\SOFTWARE\Design Science\DSMT4\WordCommands"
'Registry entry for MathType's Word Commands data

Public Const mtreg_MT_WORD_CONVFROM As String = "ConvertFrom"    'ConvertFrom
key
Public Const mtreg_MT_WORD_CONVTO As String = "ConvertTo"    'ConvertTo key
Public Const mtreg_MT_WORD_CONVMISC As String = "ConvertMisc"    'ConvertMisc key
Public Const mtreg_MT_WORD_CONVTRANS As String = "ConvertTranslator"
'ConvertTranslator key

Public Const mtreg_MT_WORD_DONTSHOW_EQNREFDLG As String =
"NoInsertEqnRefDlg"    'Don't Show Insert Eqn Ref dialog key
Public Const mtreg_MT_WORD_DONTSHOW_SLOWEQNUPDATE As String =
"NoSlowUpdateEqnDlg"    'Don't Show Insert Eqn Ref dialog key

Public Const mtreg_MT_WORD_DONTSHOW_LANGDLLERROR As String =
"NoLanuageDLLError"    'Don't show Missing Lang DLL error key

'----------- Strings used in MT text equations (TeX and MathML) ---------------------------
Public Const mttexteqn_START As String = "% MathType!"    'The identifier at the beginning
of MathType translator text equations
Public Const mttexteqn_END As String = "% MathType!End!"    'The identifier at the end of
MathType translator text equations

'----------- Property names ---------------------------
Public Const mtprop_USE_MATHTYPE_PREFS  As String = "MTUseMTPrefs"        'The
name of the Document Property that indicates to use MathType's prefs for new equations
Public Const mtprop_PREFERENCES        As String = "MTPreferences"        'Contains the
doc's settings for new equations
Public Const mtprop_PREFERENCES_FILE    As String = "MTPreferenceSource"    'Contains
the doc's settings for new equations
Public Const mtprop_NUMBER_PREFS        As String = "MTEquationNumber"     'Contains
the current equation number format preferences
Public Const mtprop_DEFER_FIELD_UPDATE  As String = "MTDeferFieldUpdate"
```

```
'Controls field updating
Public Const mtpropEQUATION_SECTION_CHECKED As String = "MTEquationSection"
'Indicates if eqn section number is 0 check has been made
Public Const mtprop_EQNREFPANE As String = "MTEqnRefPane" 'Pane number containing
insertion point where ref. is to be placed

'---------- AutoText entry names ---------------------------
Public Const mtautotext_MT3_EQN_NUMBER_FORMAT As String =
"ZMTEqnNumFormatPrefs" 'The name of old Autotext entry that held MathType3's equation
number format prototype

'---------- MathType OLE data ---------------------------
Public Const mtole_PROGID As String = "Equation.DSMT4"  'OLE Prog ID used to identify
MathType 4

'---------- Style names ---------------------------
Public Const mtstyle_EQUATION_SECTION As String = "MTEquationSection"  'Style used for
eqn. section names
Public Const mtstyle_DISPLAY_EQUATION As String = "MTDisplayEquation"  'Style used
for display equations

'---------- Misc. constants ---------------------------
'Constants used to specify 'curent selection' or 'whole document'
Public Const mt_RANGE_DOCUMENT = 0
Public Const mt_RANGE_SELECTION = 1

'Constants used by MTMsgBox
Public Const mt_MBYESNO = 1
Public Const mt_MBYESNOCANCEL = 2
Public Const mt_MBYES = 1
Public Const mt_MBNO = 2
Public Const mt_MBCANCEL = 3

'Flag bit for MTLib.SaveWordState()
Public Const mt_SWS_TRACKCHANGES = 1
Public Const mt_SWS_SMART_CUTPASTE = 2
Public Const mt_SWS_TYPING_REPLACE_SELECTION = 4
```

```
' MTUtil.bas
Attribute VB_Name = "MTUtil"
'MTUtil: 4.0
'=========================================================================
' (c) Copyright 1992-1999 by Design Science, Inc. All rights reserved
' with the exception that registered MathType owners may alter these
' macros for use by themselves and other registered MathType owners
' provided that:
'  1) The alterations are summarized in a comment directly below this
'     copyright notice. The comment should start with the words
'     "Modified by" and include the name of the person altering the
'     macros, the date of alteration, and that person's email address
'     (if available).
'  2) Persons altering the macros notify Design Science of the nature
'     of any changes they have made.
' These provisions may help us help other customers, and will help us
' continue to provide quality products for you in the future.
'=========================================================================

'This macro contains subroutines used by other Design Science macros
'=========================================================================
Option Explicit

'Public Sub Main()
'   MsgBox MTUtil.GetUserString("!1600This contains a library of functions shared by
MathType's macros."), _
'       vbOKOnly, MTUtil.GetUserString("!1601MTUtil Macro")
'End Sub

'=========================================================================
'                   CheckMTDLLVersion()
'Checks the MT DLL version. If it's a bad version, we display an
'error and return 0. If we can still run, returns nonzero
'=========================================================================
Public Function CheckMTDLLVersion()
   Dim errorflag
   Dim dllver
   Dim msg$
   Dim myResult

   errorflag = 0
   CheckMTDLLVersion = 1   'assume success to start

   'init the API
```

```
    If MTInitAPI(mtinitLAUNCH_AS_NEEDED, 30) <> 0 Then
        msg$ = MTUtil.GetUserString("!1606The MathType commands could not communicate
with MathType. There was a problem starting the API. Please be sure that MathType is properly
installed.")
        CheckMTDLLVersion = 0
        errorflag = 1
    Else
        'get the API Version
        dllver = MTAPIVersion(MTAPI_VERSION)

        'check the version against our constants
        If (dllver > mtversMajVerHi) Or (dllver < mtversMajVerLo) Then
            msg$ = MTUtil.GetUserString("!1607The version of this macro doesn't match the
version of MathType's DLL. Reinstall MathType to fix this condition.")
            CheckMTDLLVersion = 0
            errorflag = 1
        ElseIf (dllver < mtversMinVer) Then
            msg$ = MTUtil.GetUserString("!1608A more recent version of MathType's DLL is
required to use this macro. Reinstall MathType to fix this condition.")
            CheckMTDLLVersion = 0
            errorflag = 1
        End If
    End If

    If (errorflag = 1) Then     'report error condition
        MsgBox msg$, vbCritical, MTUtil.GetUserString("!1609MathType Commands for
Microsoft Word Error")
    End If
End Function

'==================================================================
'               GetUserString$
'==================================================================
Public Function GetUserString$(EnglishString$)
    'simply return the English version (strip "!nnnn" from start)
    GetUserString$ = right(EnglishString$, Len(EnglishString$) - 5)
End Function


'==================================================================
'                   GetMathTypeDir$
'   Gets the location of MathType from the registry
'==================================================================
Public Function GetMathTypeDir$()
    Dim path$
```

```
   'get the location of Mathtype from the registry
   path$ = System.PrivateProfileString("", mtreg_MT_PROGDIR_LOCATION,
mtreg_MT_PROGDIR_KEY)

   'return the results
   GetMathTypeDir$ = path$
End Function


'=====================================================================
'              WritePermSetting
'=====================================================================
'Writes key/value pair to permanent location, ie Windows registry.
'Used when data needs to be saved whose scope is larger than a document.
Public Sub WritePermSetting(key$, data$)
   System.PrivateProfileString("", mtreg_MT_WORDCMDS_LOCATION, key$) = data$
End Sub

'=====================================================================
'              ReadPermSetting$
'=====================================================================
'Reads key's value from the permanent location, ie Windows registry.
'Used when data needs to be saved whose scope is larger than a document.
Public Function ReadPermSetting$(key$)
   ReadPermSetting$ = System.PrivateProfileString("", mtreg_MT_WORDCMDS_LOCATION,
key$)
End Function
'=====================================================================
'                 SetNextTXFormPrefs
'Sets prefs that MathType will use for the next transformed equation.
'Returns MTXFormSetPrefs result code.
'=====================================================================
Function SetNextTXFormPrefs(prefStr$)
   Dim stat

   'set preferences for next transformed equation
   stat = MTXFormSetPrefs(mtxfmPREF_USER, prefStr$)

   If stat <> 0 Then
      MsgBox MTUtil.GetUserString("!1100There was a problem sending your equation
preferences for " _
         + "this document to MathType. This equation will use MathType's " _
         + "'New Equation' preferences."), vbExclamation, _
         MTUtil.GetUserString("!1101MathType Preferences Problem")
   End If
   SetNextTXFormPrefs = stat
End Function

'=====================================================================
'              SetPrefsForNextEqn
'Sets prefs that MathType will use for the next new equation.
'Returns MTSetMTPrefs result code.
'=====================================================================
Public Function SetPrefsForNextEqn(prefStr$, inline As Boolean)
   Dim stat
   Dim options As Integer

   options = mtprfMODE_NEXT_EQN
   If inline Then options = options + mtprfMODE_INLINE
   'set preferences for next transformed equation
   stat = MTSetMTPrefs(options, prefStr$, -1)
   If stat <> 0 Then
      MsgBox MTUtil.GetUserString("!1100There was a problem sending your equation
preferences for " _
         + "this document to MathType. This equation will use MathType's " _
         + "'New Equation' preferences."), vbExclamation, _
         MTUtil.GetUserString("!1101MathType Preferences Problem")
   End If
   SetPrefsForNextEqn = stat
End Function
'=====================================================================
'              IsEquationProgID
'Returns 1 if the progID is a MathType/EE OLE1 progID.
'Returns 2 if the progID is a MathType/EE OLE2 progID.
'Returns 0 if not a recognized progID.
'=====================================================================
Public Function IsEquationProgID(progID$) As Long
   Dim uProgID$
   uProgID$ = UCase(progID$)

   If uProgID$ = "EQUATION" Then
      IsEquationProgID = 1
   ElseIf InStr(1, uProgID$, "EQUATION.", vbBinaryCompare) = 1 Then
      IsEquationProgID = 2
   Else
      IsEquationProgID = 0
   End If
End Function

'=====================================================================
'            TransformGraphicEquation
'Attempts to transform the graphic on the clipboard into an equation.
'Resulting format depends on how MathType has been configured by a
'previous call to MTXFormSetTranslator.
'The transformed equation is left on the clipboard.
'If OK, returns mtOK
'If not an equation, or an error occurred, returns mtNOT_EQUATION
'==========================================================================
Public Function TransformGraphicEquation() As Long
  TransformGraphicEquation = mtNOT_EQUATION

  'Use API call to check clipboard contents first
  If MTEquationOnClipboard() = mtNOT_EQUATION Then
    Exit Function
  End If

  TransformGraphicEquation = TransformEquation()
End Function
'==========================================================================
'            TransformEquation
'Attempts to transform the item on the clipboard into an equation.
'Resulting format depends on how MathType has been configured by a
'previous call to MTXFormSetTranslator.
'The transformed equation is left on the clipboard.
'If OK, returns mtOK
'If not an equation, or an error occurred, returns mtNOT_EQUATION
'==========================================================================
Public Function TransformEquation() As Long
  Dim stat As Long
  Dim dummyStr1$, dummyStr2$
  Dim dummyDims As MTAPI_DIMS

  On Error GoTo err

  stat = mtNOT_EQUATION

  'as long as everything's OK, update the equation
  'set aside some buffers
  dummyStr1$ = Space(1)
  dummyStr2$ = Space(1)
  With dummyDims
    .baseline = 0
    .bounds.bottom = 0
    .bounds.left = 0
    .bounds.right = 0
```

```
        .bounds.top = 0
    End With

    'do the update
    stat = MTXFormEqn(mtxfmCLIPBOARD, mtxfmTEXT, dummyStr1$, 1, _
        mtxfmCLIPBOARD, mtxfmTEXT, dummyStr2$, 1, dummyDims)
    If stat < 0 Then
        stat = mtNOT_EQUATION
    End If
    GoTo Bye

err:
    If err.Number = 5690 Or err.Number = 4198 Then
        'the user has revisions on, and this is an old revision that has been deleted
        stat = -2
        Resume Bye
    Else
        err.Raise err.Number
        Stop
    End If
Bye:
    TransformEquation = stat
End Function


'=====================================================================
'               DeleteDocProperty
'=====================================================================
'deletes document property, OK to call if it doesn't exist
Public Function DeleteDocProperty(doc As Document, prop$)
    On Error GoTo Error
    doc.CustomDocumentProperties(prop$).Delete
Error:
End Function

'=====================================================================
'               DocPropertyExists
'=====================================================================
'returns True if the active document contains the custom doc property
Public Function DocPropertyExists(propName$) As Boolean
    Dim name$

    DocPropertyExists = False
    On Error GoTo Error
    name$ = ActiveDocument.CustomDocumentProperties(propName$).name
```

```
   DocPropertyExists = True
Error:
End Function


'=====================================================================
'            Delay
'Pauses execution for timeout (in milliSecs)
'=====================================================================
Public Sub Delay(timeout As Long)
   Dim start As Long
   start = GetTickCount()
   Do While (GetTickCount() < (start + timeout))
      DoEvents    ' Yield to other processes.
   Loop
End Sub
→
```

09585456.040700

```
' Timer.bas
Attribute VB_Name = "Module1"
Option Explicit

Declare Function SetTimer Lib "user32" (ByVal hWnd As Long, _
   ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerProc As Long) _
   As Long

Declare Function KillTimer Lib "user32" (ByVal hWnd As Long, _
   ByVal nIDEvent As Long) As Long

Public gProlog As Prolog
Public gTimerID As Long


' called by SolveConstraintsRandomly in Prolog.cls
Public Sub SolveAsync()

   ' calls TimerCallback when timer runs out (it's set for 0, so it
   ' runs out immediately.  TimerCallback, and anything called by
   ' TimerCallback, run async.
   gTimerID = SetTimer(0, 0, 1000, AddressOf TimerCallback)

End Sub

Public Sub TimerCallback(ByVal hWnd As Long, ByVal uMsg As Long, ByVal idEvent As
Long, ByVal dwTime As Long)

   KillTimer 0, gTimerID

   gProlog.SolveConstraintsAsync ' in Prolog.cls


End Sub
```

```
' Contraint.frm
VERSION 5.00
Object = "{BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0"; "TABCTL32.OCX"
Begin VB.Form frmConstraints
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "Create or Change Constraints"
  ClientHeight   =  6405
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  6285
  LinkTopic      =  "Form1"
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  6405
  ScaleWidth     =  6285
  ShowInTaskbar  =  0  'False
  StartUpPosition =  1  'CenterOwner
  Begin TabDlg.SSTab sstConstraintTool
    Height         =  3375
    Left           =  240
    TabIndex       =  5
    Top            =  1080
    Width          =  4455
    _ExtentX       =  7858
    _ExtentY       =  5953
    _Version       =  393216
    TabHeight      =  520
    BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
      Name           =  "MS Sans Serif"
      Size           =  8.25
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    TabCaption(0)  =  "Operators"
    TabPicture(0)  =  "Constraint.frx":0000
    Tab(0).ControlEnabled=  -1  'True
    Tab(0).Control(0)=  "cmdElseIf"
    Tab(0).Control(0).Enabled=  0  'False
    Tab(0).Control(1)=  "cmdElse"
    Tab(0).Control(1).Enabled=  0  'False
    Tab(0).Control(2)=  "cmdThen"
Tab(0).Control(2).Enabled=  0  'False
Tab(0).Control(3)=  "cmdIf"
Tab(0).Control(3).Enabled=  0  'False
Tab(0).Control(4)=  "cmdLessThanOrEqualTo"
Tab(0).Control(4).Enabled=  0  'False
Tab(0).Control(5)=  "cmdGreaterThanEqualTo"
Tab(0).Control(5).Enabled=  0  'False
Tab(0).Control(6)=  "cmdLessThan"
Tab(0).Control(6).Enabled=  0  'False
Tab(0).Control(7)=  "cmdGreaterThan"
Tab(0).Control(7).Enabled=  0  'False
Tab(0).Control(8)=  "cmdNotEqual"
Tab(0).Control(8).Enabled=  0  'False
Tab(0).Control(9)=  "cmdAbs"
Tab(0).Control(9).Enabled=  0  'False
Tab(0).Control(10)=  "cmdFactorial"
Tab(0).Control(10).Enabled=  0  'False
Tab(0).Control(11)=  "cmdExponent"
Tab(0).Control(11).Enabled=  0  'False
Tab(0).Control(12)=  "cmdQuotient"
Tab(0).Control(12).Enabled=  0  'False
Tab(0).Control(13)=  "cmdList"
Tab(0).Control(13).Enabled=  0  'False
Tab(0).Control(14)=  "cmdModulus"
Tab(0).Control(14).Enabled=  0  'False
Tab(0).Control(15)=  "cmdEqual"
Tab(0).Control(15).Enabled=  0  'False
Tab(0).Control(16)=  "cmdDivide"
Tab(0).Control(16).Enabled=  0  'False
Tab(0).Control(17)=  "cmdMultiply"
Tab(0).Control(17).Enabled=  0  'False
Tab(0).Control(18)=  "cmdMinus"
Tab(0).Control(18).Enabled=  0  'False
Tab(0).Control(19)=  "cmdPlus"
Tab(0).Control(19).Enabled=  0  'False
Tab(0).Control(20)=  "cmdParens"
Tab(0).Control(20).Enabled=  0  'False
Tab(0).ControlCount=  21
TabCaption(1)  =  "Variables"
TabPicture(1)  =  "Constraint.frx":001C
Tab(1).ControlEnabled=  0  'False
Tab(1).Control(0)=  "cboVariableNames"
Tab(1).Control(0).Enabled=  0  'False
Tab(1).Control(1)=  "cmdInsertVN"
Tab(1).Control(1).Enabled=  0  'False
Tab(1).ControlCount=  2
TabCaption(2)  =  "Functions"
TabPicture(2)  =  "Constraint.frx":0038
Tab(2).ControlEnabled=  0  'False
Tab(2).Control(0)=  "cboFunction"
Tab(2).Control(0).Enabled=  0  'False
Tab(2).Control(1)=  "cmdInsertFunction"
Tab(2).Control(1).Enabled=  0  'False
Tab(2).Control(2)=  "txtFunctionDescription"
Tab(2).Control(2).Enabled=  0  'False
Tab(2).ControlCount=  3
Begin VB.CommandButton cmdParens
  Caption        =  "( )"
  BeginProperty Font
    Name           =  "MS Sans Serif"
    Size           =  9.75
    Charset        =  0
    Weight         =  400
    Underline      =  0  'False
    Italic         =  0  'False
    Strikethrough  =  0  'False
  EndProperty
  Height         =  375
  Left           =  2280
  TabIndex       =  32
  ToolTipText    =  "List"
  Top            =  1320
  Width          =  495
End
Begin VB.ComboBox cboFunction
  Height         =  315
  ItemData       =  "Constraint.frx":0054
  Left           =  -74400
  List           =  "Constraint.frx":007C
  Style          =  2  'Dropdown List
  TabIndex       =  31
  ToolTipText    =  "Select a Prolog function from the list."
  Top            =  840
  Width          =  2175
End
Begin VB.CommandButton cmdInsertFunction
  Caption        =  "Insert"
  Height         =  315
  Left           =  -72120
  TabIndex       =  30
```

```
    ToolTipText    =  "Click here to insert this function into the constraint above at the current
cursor position."
    Top          =  840
    Width        =  855
  End
  Begin VB.TextBox txtFunctionDescription
    Height       =  1455
    Left         =  -74400
    Locked       =  -1  'True
    MultiLine    =  -1  'True
    ScrollBars   =  2  'Vertical
    TabIndex     =  29
    ToolTipText    =  "The description of the function appears in this window."
    Top          =  1320
    Width        =  3135
  End
  Begin VB.ComboBox cboVariableNames
    Height       =  315
    ItemData     =  "Constraint.frx":00EF
    Left         =  -74400
    List         =  "Constraint.frx":0117
    Style        =  2  'Dropdown List
    TabIndex     =  28
    ToolTipText    =  "Select a Prolog function from the list."
    Top          =  1320
    Width        =  2175
  End
  Begin VB.CommandButton cmdInsertVN
    Caption      =  "Insert"
    Height       =  315
    Left         =  -72120
    TabIndex     =  27
    ToolTipText    =  "Click here to insert this variable name into the constraint above at the
current cursor position."
    Top          =  1320
    Width        =  855
  End
  Begin VB.CommandButton cmdPlus
    Caption      =  "+"
    BeginProperty Font
      Name         =  "MS Sans Serif"
      Size         =  9.75
      Charset      =  0
      Weight       =  400
      Underline    =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height          =  375
    Left            =  480
    TabIndex        =  25
    ToolTipText     =  "Plus"
    Top             =  840
    Width           =  495
  End
  Begin VB.CommandButton cmdMinus
    Caption         =  "-"
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height          =  375
    Left            =  1080
    TabIndex        =  24
    ToolTipText     =  "Minus"
    Top             =  840
    Width           =  495
  End
  Begin VB.CommandButton cmdMultiply
    Caption         =  "*"
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height          =  375
    Left            =  1680
    TabIndex        =  23
    ToolTipText     =  "Multiply"
    Top             =  840
    Width           =  495
    End
    Begin VB.CommandButton cmdDivide
      Caption         =   "/"
      BeginProperty Font
        Name            =   "MS Sans Serif"
        Size            =   9.75
        Charset         =   0
        Weight          =   400
        Underline       =   0   'False
        Italic          =   0   'False
        Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   2280
      TabIndex        =   22
      ToolTipText     =   "Divide"
      Top             =   840
      Width           =   495
    End
    Begin VB.CommandButton cmdEqual
      Caption         =   "="
      BeginProperty Font
        Name            =   "MS Sans Serif"
        Size            =   9.75
        Charset         =   0
        Weight          =   400
        Underline       =   0   'False
        Italic          =   0   'False
        Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   480
      TabIndex        =   21
      ToolTipText     =   "Equals"
      Top             =   1800
      Width           =   495
    End
    Begin VB.CommandButton cmdModulus
      Caption         =   "%"
      BeginProperty Font
        Name            =   "MS Sans Serif"
        Size            =   9.75
        Charset         =   0
        Weight          =   400
        Underline       =   0   'False
    Italic        =  0  'False
    Strikethrough  =  0  'False
  EndProperty
  Height        =  375
  Left        =  2880
  TabIndex      =  20
  ToolTipText    =  "Modulo"
  Top         =  840
  Width        =  495
End
Begin VB.CommandButton cmdList
  Caption       =  "([1,2])"
  BeginProperty Font
    Name         =  "MS Sans Serif"
    Size        =  9.75
    Charset       =  0
    Weight        =  400
    Underline      =  0  'False
    Italic        =  0  'False
    Strikethrough  =  0  'False
  EndProperty
  Height        =  375
  Left        =  2880
  TabIndex      =  19
  ToolTipText    =  "List"
  Top         =  1320
  Width        =  1095
End
Begin VB.CommandButton cmdQuotient
  Caption       =  "\"
  BeginProperty Font
    Name         =  "MS Sans Serif"
    Size        =  9.75
    Charset       =  0
    Weight        =  400
    Underline      =  0  'False
    Italic        =  0  'False
    Strikethrough  =  0  'False
  EndProperty
  Height        =  375
  Left        =  480
  TabIndex      =  18
  ToolTipText    =  "Quotient"
  Top         =  1320
  Width        =  495
```

```
End
Begin VB.CommandButton cmdExponent
  Caption         =   "^"
  BeginProperty Font
    Name            =   "MS Sans Serif"
    Size            =   9.75
    Charset         =   0
    Weight          =   400
    Underline       =   0   'False
    Italic          =   0   'False
    Strikethrough   =   0   'False
  EndProperty
  Height          =   375
  Left            =   3480
  TabIndex        =   17
  ToolTipText     =   "Exponent"
  Top             =   840
  Width           =   495
End
Begin VB.CommandButton cmdFactorial
  Caption         =   "!"
  BeginProperty Font
    Name            =   "MS Sans Serif"
    Size            =   9.75
    Charset         =   0
    Weight          =   400
    Underline       =   0   'False
    Italic          =   0   'False
    Strikethrough   =   0   'False
  EndProperty
  Height          =   375
  Left            =   1080
  TabIndex        =   16
  ToolTipText     =   "Factorial"
  Top             =   1320
  Width           =   495
End
Begin VB.CommandButton cmdAbs
  Caption         =   "| |"
  BeginProperty Font
    Name            =   "MS Sans Serif"
    Size            =   9.75
    Charset         =   0
    Weight          =   400
    Underline       =   0   'False
```

```
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height        =  375
    Left          =  1680
    TabIndex      =  15
    ToolTipText   =  "Absolute value"
    Top           =  1320
    Width         =  495
  End
  Begin VB.CommandButton cmdNotEqual
    Caption       =  "=/="
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height        =  375
    Left          =  1080
    TabIndex      =  14
    ToolTipText   =  "Does not equal"
    Top           =  1800
    Width         =  495
  End
  Begin VB.CommandButton cmdGreaterThan
    Caption       =  ">"
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height        =  375
    Left          =  1680
    TabIndex      =  13
    ToolTipText   =  "Greater than"
    Top           =  1800
    Width         =  495
```

```
      End
      Begin VB.CommandButton cmdLessThan
        Caption       =  "<"
        BeginProperty Font
          Name          =  "MS Sans Serif"
          Size          =  9.75
          Charset       =  0
          Weight        =  400
          Underline     =  0  'False
          Italic        =  0  'False
          Strikethrough =  0  'False
        EndProperty
        Height        =  375
        Left          =  2280
        TabIndex      =  12
        ToolTipText   =  "Less than"
        Top           =  1800
        Width         =  495
      End
      Begin VB.CommandButton cmdGreaterThanEqualTo
        Caption       =  ">="
        BeginProperty Font
          Name          =  "MS Sans Serif"
          Size          =  9.75
          Charset       =  0
          Weight        =  400
          Underline     =  0  'False
          Italic        =  0  'False
          Strikethrough =  0  'False
        EndProperty
        Height        =  375
        Left          =  2880
        TabIndex      =  11
        ToolTipText   =  "Greater than or equal to"
        Top           =  1800
        Width         =  495
      End
      Begin VB.CommandButton cmdLessThanOrEqualTo
        Caption       =  "<="
        BeginProperty Font
          Name          =  "MS Sans Serif"
          Size          =  9.75
          Charset       =  0
          Weight        =  400
          Underline     =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height         =  375
    Left           =  3480
    TabIndex       =  10
    ToolTipText    =  "Less than or equal to"
    Top            =  1800
    Width          =  495
  End
  Begin VB.CommandButton cmdIf
    Caption        =  "if"
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height         =  375
    Left           =  480
    TabIndex       =  9
    ToolTipText    =  "If"
    Top            =  2280
    Width          =  735
  End
  Begin VB.CommandButton cmdThen
    Caption        =  "then"
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  9.75
      Charset        =  0
      Weight         =  400
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height         =  375
    Left           =  1320
    TabIndex       =  8
    ToolTipText    =  "then"
    Top            =  2280
    Width          =  735
```

```
      End
      Begin VB.CommandButton cmdElse
        Caption         =   "else"
        BeginProperty Font
          Name            =   "MS Sans Serif"
          Size            =   9.75
          Charset         =   0
          Weight          =   400
          Underline       =   0   'False
          Italic          =   0   'False
          Strikethrough   =   0   'False
        EndProperty
        Height          =   375
        Left            =   2160
        TabIndex        =   7
        ToolTipText     =   "else"
        Top             =   2280
        Width           =   735
      End
      Begin VB.CommandButton cmdElseIf
        Caption         =   "elseif"
        BeginProperty Font
          Name            =   "MS Sans Serif"
          Size            =   9.75
          Charset         =   0
          Weight          =   400
          Underline       =   0   'False
          Italic          =   0   'False
          Strikethrough   =   0   'False
        EndProperty
        Height          =   375
        Left            =   3000
        TabIndex        =   6
        ToolTipText     =   "elseif"
        Top             =   2280
        Width           =   975
      End
    End
    Begin VB.TextBox txtConstraint
      Height          =   315
      Left            =   240
      TabIndex        =   3
      ToolTipText     =   "Enter the constraint here."
      Top             =   480
      Width           =   4455
```

```
End
Begin VB.TextBox txtComment
  Height        =  1335
  Left          =  240
  MultiLine     =  -1  'True
  TabIndex      =  0
  Top           =  4800
  Width         =  4455
End
Begin VB.CommandButton cmdConOK
  Caption       =  "OK"
  Default       =  -1  'True
  Height        =  495
  Left          =  4920
  TabIndex      =  1
  ToolTipText   =  "Click here to save this constraint."
  Top           =  120
  Width         =  1215
End
Begin VB.CommandButton cmdConCancel
  Caption       =  "Cancel"
  Height        =  495
  Left          =  4920
  TabIndex      =  2
  ToolTipText   =  "Click here to return without creating or modifying this constraint."
  Top           =  720
  Width         =  1215
End
Begin VB.Label lblComment
  Caption       =  "Comment"
  Height        =  255
  Left          =  240
  TabIndex      =  26
  Top           =  4560
  Width         =  1215
End
Begin VB.Label lblConstraints
  Caption       =  "Constraint"
  Height        =  255
  Left          =  240
  TabIndex      =  4
  ToolTipText   =  "Click on the down arrow for function prototypes"
  Top           =  240
  Width         =  1695
End
```

```
End
Attribute VB_Name = "frmConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mbytAddEditFlag As Byte

Private mlstListBox As ListBox

Private mudtCon As Constraint

Private mudtModel As Model

Private mudtConType As ConstraintType

Private Enum ResourceStrings
   rcStartFunctions = 101
   rcEndFunctions = 125
   rcStartExplanations = 201
End Enum

Private mblnChangeFocus As Boolean

Public Property Let AddEditFlag(ByVal bytNewValue As Byte)

   mbytAddEditFlag = bytNewValue

End Property

Public Property Let ListBox(ByVal lstNewValue As ListBox)

   Set mlstListBox = lstNewValue

End Property

Public Property Let Constraint(ByVal udtNewValue As Constraint)

   Set mudtCon = udtNewValue

End Property

Public Property Let ConstraintType(ByVal udtNewValue As ConstraintType)
```

```
   mudtConType = udtNewValue

End Property

Public Property Let Model(ByVal udtNewValue As Model)

   Set mudtModel = udtNewValue

End Property

Private Sub cboFunction_Click()

   Dim intI As Integer

   For intI = 0 To cboFunction.ListCount - 1
      If cboFunction = cboFunction.List(intI) Then
         txtFunctionDescription = LoadResString(intI + rcStartExplanations)
         Exit For
      End If
   Next intI

   If mblnChangeFocus Then
      txtConstraint.SetFocus
   End If

End Sub

Private Sub cboVariableNames_Click()

   If mblnChangeFocus Then
      txtConstraint.SetFocus
   End If

End Sub

Private Sub cmdElse_Click()

   Call InsertText("else", 0)

End Sub

Private Sub cmdElseIf_Click()

   Call InsertText("elseif", 0)
```

```
End Sub

Private Sub cmdGreaterThan_Click()

   Call InsertText(">", 0)

End Sub

Private Sub cmdGreaterThanEqualTo_Click()

   Call InsertText(">=", 0)

End Sub

Private Sub cmdIf_Click()

   Call InsertText("if", 0)

End Sub

Private Sub cmdParens_Click()

   Call InsertText("()", 1)

End Sub

Private Sub cmdThen_Click()

   Call InsertText("then", 0)

End Sub

Private Sub cmdInsertFunction_Click()

   If cboFunction = "brandom()" Or cboFunction = "random()" Then
      Call InsertText(cboFunction, 0)
   Else
      Call InsertText(cboFunction, 1)
   End If

End Sub

Private Sub cmdInsertVN_Click()

   Call InsertText(cboVariableNames, 0)
```

```
End Sub
Private Sub cmdLessThan_Click()
   Call InsertText("<")
End Sub
Private Sub cmdLessThanOrEqualTo_Click()
   Call InsertText("<=", 0)
End Sub
Private Sub cmdNotEqual_Click()
   Call InsertText("=/=", 0)
End Sub
Private Sub cmdPlus_Click()
   Call InsertText("+")
End Sub
Private Sub cmdMinus_Click()
   Call InsertText("-")
End Sub
Private Sub cmdMultiply_Click()
   Call InsertText("*")
End Sub
Private Sub cmdDivide_Click()
   Call InsertText("/")
End Sub
```

```
Private Sub cmdModulus_Click()
   Call InsertText("%")
End Sub
Private Sub cmdEqual_Click()
   Call InsertText("=")
End Sub
Private Sub cmdList_Click()
   Call InsertText("([])", 2)
End Sub
Private Sub cmdQuotient_Click()
   Call InsertText("\")
End Sub
Private Sub cmdExponent_Click()
   Call InsertText("^")
End Sub
Private Sub cmdFactorial_Click()
   Call InsertText("!")
End Sub
Private Sub cmdAbs_Click()
   Call InsertText("||", 1)
End Sub
Private Sub InsertText(ByVal strInsertedText As String, _
   Optional ByVal intOffset As Integer = -1)
```

```
    Dim strFront As String
    Dim strBack As String

    If intOffset = -1 Then intOffset = Len(strInsertedText) - 1

    strFront = left(txtConstraint, txtConstraint.SelStart)
    strBack = right(txtConstraint, Len(txtConstraint) - _
       txtConstraint.SelStart - txtConstraint.SelLength)

    txtConstraint = strFront & strInsertedText & strBack
    txtConstraint.SetFocus

    ' move the cursor
    txtConstraint.SelStart = Len(strFront) + Len(strInsertedText) - intOffset

End Sub

Private Sub Command3_Click()

End Sub

Private Sub Form_Load()

    ' disable OK button if changes aren't allowed
    If mudtModel.IsFrozen Then
       cmdConOK.Enabled = False
    Else
       cmdConOK.Enabled = True
    End If

    Dim udtV As Variable

    ' load variable names into combo box
    cboVariableNames.Clear
    For Each udtV In mudtModel.Variables
       Call cboVariableNames.AddItem(udtV.name)
    Next udtV

    If mbytAddEditFlag = aeEdit Then
       txtConstraint = mudtCon.ConstraintString
       txtComment = mudtCon.Comment
    End If

    'load functions into combo box
    Dim intI As Integer
    For intI = rcStartFunctions To rcEndFunctions
       cboFunction.List(intI - rcStartFunctions) = LoadResString(intI)
    Next intI

    mblnChangeFocus = False
    If cboVariableNames.ListCount > 0 Then
       cboVariableNames.ListIndex = 0
    End If
    cboFunction.ListIndex = 0
    mblnChangeFocus = True

End Sub

Private Sub cmdConOK_Click()

    If Len(txtConstraint) = 0 Then
       Call MsgBox("Null constraints are not permitted", vbExclamation, "Error")
       Exit Sub
    End If

    If mbytAddEditFlag = aeEdit Then ' we're editing an old one
       ' update the constraint with new data from the form
       Call mudtCon.Update(txtConstraint, mudtConType, txtComment)
       ' update the text in the list box
       mlstListBox.List(mlstListBox.ListIndex) = mudtCon.ConstraintString
    Else
       ' Add the new constraint
       Set mudtCon = mudtModel.Constraints.Add(txtConstraint, True, _
          mudtConType, txtComment)
       With mlstListBox
          ' Add the new constraint to the list box
          Call .AddItem(mudtCon.ConstraintString)
          ' Set ItemData to index value of the variable object
          .ItemData(.ListCount - 1) = mudtCon.index
          ' Check the check box
          .Selected(.ListCount - 1) = True
       End With
    End If

    Call frmTCA.AddUndefinedVariables(txtConstraint)

    Unload Me

End Sub
```

```
Private Sub cmdConCancel_Click()
    Unload Me
End Sub
```

```
' EditConstraint.frm
VERSION 5.00
Begin VB.Form frmEditText
  BorderStyle    =  1  'Fixed Single
  ClientHeight   =  1455
  ClientLeft     =  45
  ClientTop      =  330
  ClientWidth    =  4785
  LinkTopic      =  "Form1"
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  1455
  ScaleWidth     =  4785
  StartUpPosition =  3  'Windows Default
  Begin VB.CommandButton cmdEditTextOK
    Caption        =  "OK"
    Default        =  -1  'True
    Height         =  495
    Left           =  3360
    TabIndex       =  2
    Top            =  120
    Width          =  1215
  End
  Begin VB.CommandButton cmdEditTextnCancel
    Caption        =  "Cancel"
    Height         =  495
    Left           =  3360
    TabIndex       =  1
    Top            =  720
    Width          =  1215
  End
  Begin VB.TextBox txtEditText
    Alignment      =  2  'Center
    Height         =  375
    Left           =  240
    TabIndex       =  0
    Top            =  120
    Width          =  2895
  End
End
Attribute VB_Name = "frmEditText"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
' These are used as references to the ListBox in frmTCA currently being editted
Public lstListBox As ListBox
Public intInd As Integer

Private Sub cmdEditTextnCancel_Click()
   Unload Me
End Sub

Private Sub cmdEditTextOK_Click()
   lstListBox.AddItem txtEditText.Text
   lstListBox.RemoveItem intInd
   Unload Me
End Sub
```

```
' Form1.frm
VERSION 5.00
Begin VB.Form Form1
  Caption        =  "Form1"
  ClientHeight   =  4050
  ClientLeft     =  60
  ClientTop      =  345
  ClientWidth    =  5595
  LinkTopic      =  "Form1"
  ScaleHeight    =  4050
  ScaleWidth     =  5595
  StartUpPosition =  3  'Windows Default
  Begin VB.CommandButton Command1
    Caption        =  "Clear"
    Height         =  1455
    Left           =  3720
    TabIndex       =  2
    Top            =  2520
    Width          =  1455
  End
  Begin VB.TextBox Text1
    Height         =  855
    Left           =  600
    TabIndex       =  1
    Text           =  "Text1"
    Top            =  960
    Width          =  2175
  End
  Begin VB.CommandButton cmdRun
    Caption        =  "Run"
    Height         =  1335
    Left           =  3720
    TabIndex       =  0
    Top            =  960
    Width          =  1455
  End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
```

```
Private Sub cmdRun_Click()

   Dim udtP As New Prolog
   Dim lngR As Long

   If udtP.StartProlog("hlp4lib.p4") = False Then
      Call MsgBox("Prolog failure on startup", vbExclamation, "Error")
   End If

   Call udtP.AddVariable("int(I),[520<=I<=590 step 5], int(I2),[I + 5<=I2<=I + 30 step 1]")

   lngR = udtP.SolveConstraintsOrdered(1)

   Text1 = Str(lngR)

End Sub

Private Sub Command1_Click()

   Text1 = ""

End Sub
```

```
' frmAbout.frm
VERSION 5.00
Begin VB.Form frmAbout
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "About TCA"
  ClientHeight   =  2610
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  4440
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  2610
  ScaleWidth     =  4440
  ShowInTaskbar  =  0   'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.CommandButton cmdOK
    Caption        =  "OK"
    Height         =  495
    Left           =  3120
    TabIndex       =  1
    Top            =  120
    Width          =  1215
  End
  Begin VB.Label lblVersion
    Height         =  255
    Left           =  240
    TabIndex       =  2
    Top            =  2160
    Width          =  2295
  End
  Begin VB.Label Label1
    Caption        =  "TCA is a collaborative development of the Assessment and Research
Divisions."
    Height         =  615
    Left           =  240
    TabIndex       =  0
    Top            =  1320
    Width          =  2535
  End
  Begin VB.Image imaETS
    BorderStyle    =  1  'Fixed Single
    Height         =  780
```

```
      Left         =  960
      Picture       =  "frmAbout.frx":0000
      Top          =  240
      Width         =  1275
   End
End
Attribute VB_Name = "frmAbout"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdEasterEgg_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)

    If Button = vbRightButton Then
       ' display easter egg
       Beep
    End If

End Sub

Private Sub cmdOK_Click()

    Unload Me

End Sub

Private Sub Form_Load()

    lblVersion = frmSplash.lblVersion

End Sub

Private Sub imaETS_DblClick()

    ' display easter egg
    Beep

End Sub


' frmAttributes.frm
VERSION 5.00
```

```
Begin VB.Form frmAttributes
  BorderStyle     =  4  'Fixed ToolWindow
  Caption         =  "Family Attributes"
  ClientHeight    =  1590
  ClientLeft      =  45
  ClientTop       =  285
  ClientWidth     =  4305
  LinkTopic       =  "Form1"
  LockControls    =  -1  'True
  MaxButton       =  0   'False
  MinButton       =  0   'False
  ScaleHeight     =  1590
  ScaleWidth      =  4305
  ShowInTaskbar   =  0   'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.ComboBox cboProximity
    Height          =  315
    ItemData        =  "frmAttributes.frx":0000
    Left            =  240
    List            =  "frmAttributes.frx":000D
    Style           =  2  'Dropdown List
    TabIndex        =  4
    Top             =  360
    Width           =  1935
  End
  Begin VB.OptionButton optGeneric
    Caption         =  "Generic"
    Height          =  195
    Index           =  0
    Left            =  120
    TabIndex        =  3
    Top             =  1035
    Value           =  -1  'True
    Width           =  975
  End
  Begin VB.OptionButton optGeneric
    Caption         =  "Non-generic"
    Height          =  195
    Index           =  1
    Left            =  1080
    TabIndex        =  2
    Top             =  1035
    Width           =  1455
  End
  Begin VB.CommandButton cmdCancel
```

```
      Caption      = "Cancel"
      Height       = 495
      Left         = 3000
      TabIndex     = 1
      ToolTipText  = "Click here to return without saving these family attributes."
      Top          = 720
      Width        = 1215
   End
   Begin VB.CommandButton cmdOK
      Caption      = "OK"
      Default      = -1 'True
      Height       = 495
      Left         = 3000
      TabIndex     = 0
      ToolTipText  = "Click here to save these family attributes."
      Top          = 120
      Width        = 1215
   End
   Begin VB.Label lbl
      Caption      = "Variant proximity"
      Height       = 255
      Left         = 240
      TabIndex     = 5
      Top          = 120
      Width        = 1335
   End
End
Attribute VB_Name = "frmAttributes"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mblnOK As Boolean

Private mblnGeneric As Boolean
Private mudtProximity As Proximity

Private Sub Form_Load()

   mblnOK = False

   cboProximity.ListIndex = frmTCA.Family.Proximity
   If frmTCA.Family.Generic Then
```

```
    optGeneric(0) = True
  Else
    optGeneric(1) = True
  End If

  mblnGeneric = frmTCA.Family.Generic
  mudtProximity = frmTCA.Family.Proximity

End Sub

Public Property Get Proximity() As Proximity

  Proximity = mudtProximity

End Property

Public Property Get Generic() As Boolean

  Generic = mblnGeneric

End Property

Private Sub cmdOK_Click()

  mblnOK = True

  Unload Me

End Sub

Private Sub cmdCancel_Click()

  Unload Me

End Sub

Public Property Get OK() As Boolean

  OK = mblnOK

End Property

Private Sub cboProximity_Click()

  mudtProximity = cboProximity.ListIndex
```

```
End Sub

Private Sub optGeneric_Click(Index As Integer)

   mblnGeneric = optGeneric(0)

End Sub
```

```
' frmComments.frm
VERSION 5.00
Begin VB.Form frmComments
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "Comments"
  ClientHeight   =  3765
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  5250
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  3765
  ScaleWidth     =  5250
  ShowInTaskbar  =  0   'False
  StartUpPosition =  2  'CenterScreen
  Begin VB.CommandButton cmdCancel
    Caption        =  "Cancel"
    Height         =  495
    Left           =  3960
    TabIndex       =  2
    ToolTipText    =  "Click here to save these family attributes."
    Top            =  720
    Width          =  1215
  End
  Begin VB.CommandButton cmdOK
    Caption        =  "OK"
    Default        =  -1  'True
    Height         =  495
    Left           =  3960
    TabIndex       =  1
    ToolTipText    =  "Click here to save these family attributes."
    Top            =  120
    Width          =  1215
  End
  Begin VB.TextBox txtComment
    Height         =  3495
    Left           =  120
    MultiLine      =  -1  'True
    TabIndex       =  0
    Top            =  120
    Width          =  3735
  End
```

```
End
Attribute VB_Name = "frmComments"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private mstrComment As String

Public Property Get Comment() As String

   Comment = mstrComment

End Property

Public Property Let Comment(ByVal strNewValue As String)

   txtComment = strNewValue
   mstrComment = strNewValue

End Property

Private Sub cmdCancel_Click()

   Unload Me

End Sub

Private Sub cmdOK_Click()

   mstrComment = txtComment
   Unload Me

End Sub
```

```
' frmDifficulty.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Begin VB.Form frmDifficulty
  BorderStyle    =  4 'Fixed ToolWindow
  ClientHeight   =  8730
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  6855
  LinkTopic      =  "Form1"
  LockControls   =  -1 'True
  MaxButton      =  0 'False
  MinButton      =  0 'False
  ScaleHeight    =  8730
  ScaleWidth     =  6855
  ShowInTaskbar  =  0 'False
  StartUpPosition =  2 'CenterScreen
  Begin VB.CheckBox chkRoute
    Caption        =  "Route to TCS"
    Height         =  375
    Left           =  2640
    TabIndex       =  33
    Top            =  1800
    Width          =  1935
  End
  Begin VB.ComboBox cboKey
    Height         =  315
    ItemData       =  "frmDifficulty.frx":0000
    Left           =  2640
    List           =  "frmDifficulty.frx":0013
    Style          =  2 'Dropdown List
    TabIndex       =  30
    Top            =  1200
    Width          =  615
  End
  Begin VB.CheckBox chkCalcDifficulty
    Caption        =  "Calculate difficulty"
    Height         =  255
    Left           =  240
    TabIndex       =  27
    Top            =  3600
    Value          =  1 'Checked
    Width          =  1935
  End
```

```
Begin VB.ComboBox cboDeliveryMode
  Height        =  315
  ItemData      =  "frmDifficulty.frx":0026
  Left          =  2640
  List          =  "frmDifficulty.frx":0030
  Style         =  2  'Dropdown List
  TabIndex      =  25
  Top           =  480
  Width         =  1695
End
Begin VB.ComboBox cboDomain
  Height        =  315
  ItemData      =  "frmDifficulty.frx":003E
  Left          =  240
  List          =  "frmDifficulty.frx":004E
  Style         =  2  'Dropdown List
  TabIndex      =  18
  Top           =  1200
  Width         =  1695
End
Begin VB.OptionButton optNature
  Caption       =  "Pure"
  Height        =  375
  Index         =  0
  Left          =  240
  TabIndex      =  17
  Top           =  1800
  Value         =  -1  'True
  Width         =  735
End
Begin VB.OptionButton optNature
  Caption       =  "Real"
  Height        =  375
  Index         =  1
  Left          =  1200
  TabIndex      =  16
  Top           =  1800
  Width         =  735
End
Begin VB.CommandButton cmdOK
  Caption       =  "OK"
  Default       =  -1  'True
  Height        =  495
  Left          =  5520
  TabIndex      =  8
```

```
      ToolTipText    =  "Click here to save changes and return."
      Top         =  240
      Width        =  1215
    End
    Begin VB.CommandButton cmdCancel
      Caption       =  "Cancel"
      Height        =  495
      Left         =  5520
      TabIndex       =  7
      ToolTipText    =  "Click here to save changes and return."
      Top         =  840
      Width        =  1215
    End
    Begin VB.TextBox txtBatchId
      Height        =  315
      Left         =  240
      TabIndex       =  0
      Top         =  480
      Width        =  1695
    End
    Begin ComctlLib.Slider sldTDEstimate
      Height        =  375
      Left         =  480
      TabIndex       =  20
      Top         =  2760
      Width        =  3975
      _ExtentX       =  7011
      _ExtentY       =  661
      _Version       =  327682
      LargeChange    =  1
      Min          =  1
      Max          =  5
      SelStart       =  1
      Value         =  1
    End
    Begin VB.Frame fraPredDiff
      Caption       =  "Predicted Difficulty"
      Height        =  1575
      Left         =  480
      TabIndex       =  10
      Top         =  6720
      Width        =  4575
      Begin ComctlLib.Slider sldDiffEstimate
        Height        =  375
        Left         =  240
```

```
  TabIndex      =  11
  Top         =  720
  Width        =  3975
  _ExtentX      =  7011
  _ExtentY      =  661
  _Version      =  327682
  Min         =  1
  Max         =  5
  SelStart      =  1
  Value        =  1
End
Begin VB.Label lblIRTValue
  Height       =  255
  Left        =  1080
  TabIndex      =  32
  Top         =  360
  Width        =  3015
End
Begin VB.Label lblPredEasy
  Caption       =  "Easy"
  Height       =  255
  Left        =  3840
  TabIndex      =  15
  Top         =  1200
  Width        =  615
End
Begin VB.Label lblPredMed
  Caption       =  "Medium"
  Height       =  255
  Left        =  1920
  TabIndex      =  14
  Top         =  1200
  Width        =  855
End
Begin VB.Label lblPredDiff
  Caption       =  "Difficult"
  Height       =  255
  Left        =  240
  TabIndex      =  13
  Top         =  1200
  Width        =  735
End
Begin VB.Label lblIRT
  Caption       =  "IRT b:"
  Height       =  255
```

```
      Left          =  360
      TabIndex      =  12
      Top           =  360
      Width         =  495
    End
  End
  Begin VB.Frame fraGREDiff
    Caption         =  "GRE Difficulty "
    Height          =  4575
    Left            =  240
    TabIndex        =  2
    Top             =  3960
    Width           =  5055
    Begin VB.ComboBox cboGREConcept
      Height          =  315
      ItemData        =  "frmDifficulty.frx":0080
      Left            =  240
      List            =  "frmDifficulty.frx":0093
      Style           =  2  'Dropdown List
      TabIndex        =  28
      Top             =  2160
      Width           =  2055
    End
    Begin VB.ComboBox cboGRECog
      Height          =  315
      ItemData        =  "frmDifficulty.frx":00ED
      Left            =  240
      List            =  "frmDifficulty.frx":00FA
      Style           =  2  'Dropdown List
      TabIndex        =  5
      Top             =  1440
      Width           =  2055
    End
    Begin VB.ComboBox cboGREComp
      Height          =  315
      ItemData        =  "frmDifficulty.frx":012D
      Left            =  240
      List            =  "frmDifficulty.frx":013D
      Style           =  2  'Dropdown List
      TabIndex        =  3
      Top             =  720
      Width           =  2055
    End
    Begin VB.Label lblConcept
      Caption         =  "Concept:"
```

```
            Height        =  255
            Left          =  240
            TabIndex      =  29
            Top           =  1920
            Width         =  975
         End
         Begin VB.Label lblGRECog
            Caption       =  "Cognition:"
            Height        =  255
            Left          =  240
            TabIndex      =  6
            Top           =  1200
            Width         =  975
         End
         Begin VB.Label lblGREComp
            Caption       =  "Computation:"
            Height        =  255
            Left          =  240
            TabIndex      =  4
            Top           =  480
            Width         =  975
         End
      End
      Begin VB.Frame fraGMATDiff
         Caption       =  "GMAT Difficulty"
         Height        =  4575
         Left          =  240
         TabIndex      =  9
         Top           =  3960
         Width         =  5055
      End
      Begin VB.Frame fraOther
         Height        =  4575
         Left          =  240
         TabIndex      =  34
         Top           =  3960
         Width         =  5055
      End
      Begin VB.Label lblKey
         Caption       =  "Key:"
         Height        =  255
         Left          =  2640
         TabIndex      =  31
         Top           =  960
         Width         =  975
```

```
End
Begin VB.Label lblTarget
  Caption       =  "Target template:"
  Height        =  255
  Left        =  2640
  TabIndex      =  26
  Top         =  240
  Width        =  1815
End
Begin VB.Label lblSlideDirections
  Caption       =  "Adjust the slide to estimated variant difficulty:"
  Height        =  255
  Left        =  600
  TabIndex      =  24
  Top         =  2400
  Width        =  3615
End
Begin VB.Label lblTDDiff
  Caption       =  "Difficult"
  Height        =  255
  Left        =  480
  TabIndex      =  23
  Top         =  3240
  Width        =  735
End
Begin VB.Label lblTDMed
  Caption       =  "Medium"
  Height        =  255
  Left        =  2160
  TabIndex      =  22
  Top         =  3240
  Width        =  855
End
Begin VB.Label lblTDEasy
  Caption       =  "Easy"
  Height        =  255
  Left        =  4080
  TabIndex      =  21
  Top         =  3240
  Width        =  615
End
Begin VB.Label lblDomain
  Caption       =  "Domain:"
  Height        =  255
  Left        =  240
```

```
      TabIndex      =  19
      Top           =  960
      Width         =  975
    End
    Begin VB.Label LblBatch
      Caption       =  "Batch id:"
      Height        =  255
      Left          =  240
      TabIndex      =  1
      Top           =  240
      Width         =  975
    End
  End
  Attribute VB_Name = "frmDifficulty"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = True
  Attribute VB_Exposed = False
  Option Explicit

  Dim mudtFamily As Family
  Dim mudtClone As Clone
  Dim mudtDE As DifficultyEstimate
  Dim mudtGreDE As GREDifficultyEstimate
  Dim mudtGmatDE As GMATDifficultyEstimate

  Dim mblnFormLoad As Boolean

  Public Property Let Family(ByVal udtNewValue As Family)

      Set mudtFamily = udtNewValue

  End Property

  Public Property Let Clone(ByVal udtNewValue As Clone)

      Set mudtClone = udtNewValue

  End Property

  Private Sub Form_Load()

      Set mudtDE = mudtClone.DiffEst

      mblnFormLoad = True
```

```
' if there's a key, prohibit input.
If mudtFamily.ItemType = ptStandardMC Then
   cboKey.Enabled = False
Else
   cboKey.Enabled = True
End If

' change form depending on program
Select Case mudtFamily.Program
   Case prGRE
      fraGREDiff.ZOrder
      fraPredDiff.ZOrder
   Case prGMAT
      fraGMATDiff.ZOrder
      fraPredDiff.ZOrder
   Case Else
      fraOther.ZOrder
End Select

cboDomain.ListIndex = mudtClone.Domain
txtBatchId = mudtClone.BatchID
cboDeliveryMode.ListIndex = mudtClone.DeliveryMode

' if key is not set, force "A"
If mudtClone.key = "" Then
   cboKey = "A"
Else
   cboKey = mudtClone.key
End If

If mudtClone.Nature = naPure Then
   optNature(0) = True
Else
   optNature(1) = True
End If

sldTDEstimate = mudtClone.TDEstimate
chkRoute = mudtClone.IsRouted
chkCalcDifficulty = mudtClone.IsDifficultyCalculated
chkCalcDifficulty_Click ' update screen accordingly
If mudtClone.IsDifficultyCalculated Then
   Select Case mudtFamily.Program
      Case prGRE
         Set mudtGreDE = mudtClone.DiffEst
         cboGREComp.ListIndex = mudtGreDE.Computation
```

```
                cboGRECog.ListIndex = mudtGreDE.Cognition
                cboGREConcept.ListIndex = mudtGreDE.Concept
                CreateDiffEst
            Case prGMAT
                Set mudtGmatDE = mudtClone.DiffEst
                ' nothing to load
                CreateDiffEst
            Case prSAT
                ' do nothing
        End Select
    Else
        cboGREComp.ListIndex = 0
        cboGRECog.ListIndex = 0
        cboGREConcept.ListIndex = 0
    End If

    mblnFormLoad = False

End Sub

Private Sub cmdOK_Click()

    CreateProfile

    Unload Me

End Sub

Private Sub cmdCancel_Click()

    Unload Me

End Sub

Private Sub cboDomain_Click()

    CreateProfile

End Sub

Private Sub cboGRECog_Click()

    CreateProfile

End Sub
```

```
Private Sub cboGREComp_Click()

   CreateProfile

End Sub

Private Sub cboGREConcept_Click()

   CreateProfile

End Sub

Private Sub cboKey_Click()

   CreateProfile

End Sub

Private Sub optNature_Click(Index As Integer)

   CreateProfile

End Sub

Private Sub sldTDEstimate_Click()

   CreateProfile

End Sub

Private Sub chkCalcDifficulty_Click()

   fraPredDiff.Enabled = CBool(chkCalcDifficulty)
   fraGREDiff.Enabled = CBool(chkCalcDifficulty)
   fraGMATDiff.Enabled = CBool(chkCalcDifficulty)
   lblGREComp.Enabled = CBool(chkCalcDifficulty)
   cboGREComp.Enabled = CBool(chkCalcDifficulty)
   lblGRECog.Enabled = CBool(chkCalcDifficulty)
   cboGRECog.Enabled = CBool(chkCalcDifficulty)
   lblConcept.Enabled = CBool(chkCalcDifficulty)
   cboGREConcept.Enabled = CBool(chkCalcDifficulty)
   lblIRT.Enabled = CBool(chkCalcDifficulty)
   lblIRTValue.Enabled = CBool(chkCalcDifficulty)
   lblPredDiff.Enabled = CBool(chkCalcDifficulty)
   lblPredEasy.Enabled = CBool(chkCalcDifficulty)
```

```
    lblPredMed.Enabled = CBool(chkCalcDifficulty)
    lblPredDiff.Enabled = CBool(chkCalcDifficulty)

    If chkCalcDifficulty Then
        CreateProfile
    End If

End Sub

Private Sub CreateProfile()

    ' don't do it if were still loading form
    If mblnFormLoad Then Exit Sub

    mudtClone.Program = mudtFamily.Program
    mudtClone.Domain = cboDomain.ListIndex
    mudtClone.BatchID = txtBatchId
    mudtClone.DeliveryMode = cboDeliveryMode.ListIndex
    mudtClone.key = cboKey
    If optNature(0) = True Then
        mudtClone.Nature = naPure
    Else
        mudtClone.Nature = naReal
    End If
    mudtClone.IsRouted = chkRoute
    mudtClone.TDEstimate = sldTDEstimate

    mudtClone.IsDifficultyCalculated = chkCalcDifficulty

    If chkCalcDifficulty Then
        CreateDiffEst
    End If

End Sub

Private Sub CreateDiffEst()

    If mudtClone.IsDifficultyCalculated Then
        Set mudtDE = Nothing
        Select Case mudtFamily.Program
            Case prGRE
                Set mudtGreDE = Nothing
                Set mudtGreDE = New GREDifficultyEstimate
                mudtGreDE.Domain = cboDomain.ListIndex
                mudtGreDE.Computation = cboGREComp.ListIndex
            mudtGreDE.Cognition = cboGRECog.ListIndex
            mudtGreDE.Concept = cboGREConcept.ListIndex
            mudtGreDE.key = cboKey
            If optNature(0) = True Then
               mudtGreDE.Nature = naPure
            Else
               mudtGreDE.Nature = naReal
            End If
            mudtGreDE.ItemType = mudtFamily.ItemType
            ' attach this GRE DE to the clone
            mudtClone.DiffEst = mudtGreDE
            Set mudtDE = mudtGreDE
            SetPredDiffSlider
         Case prGMAT
            Set mudtGmatDE = Nothing
            Set mudtGmatDE = New GMATDifficultyEstimate
            mudtGmatDE.Domain = cboDomain.ListIndex
            mudtGmatDE.key = cboKey
            If optNature(0) = True Then
               mudtGmatDE.Nature = naPure
            Else
               mudtGmatDE.Nature = naReal
            End If
            mudtGmatDE.ItemType = mudtFamily.ItemType
            mudtGmatDE.TDDiffEst = sldTDEstimate
            ' attach this GMAT DE to the clone
            mudtClone.DiffEst = mudtGmatDE
            Set mudtDE = mudtGmatDE
            SetPredDiffSlider
         Case prSAT
            ' do nothing
      End Select
   Else ' opted not to calc difficulty
      mudtClone.DiffEst = Nothing
   End If

End Sub

Private Sub SetPredDiffSlider()

   Dim dblIRT As Double

   dblIRT = mudtDE.ComputeDifficulty

   lblIRTValue = Format(dblIRT, "0.#")
```

```
        Select Case mudtFamily.Program
            Case prGRE
                If dblIRT < -1.001 Then
                    sldDiffEstimate = 5
                ElseIf dblIRT < -0.238 Then
                    sldDiffEstimate = 4
                ElseIf dblIRT < 0.379 Then
                    sldDiffEstimate = 3
                ElseIf dblIRT < 0.931 Then
                    sldDiffEstimate = 2
                Else
                    sldDiffEstimate = 1
                End If
            Case prGMAT
                If dblIRT < -0.919 Then
                    sldDiffEstimate = 5
                ElseIf dblIRT < -0.093 Then
                    sldDiffEstimate = 4
                ElseIf dblIRT < 0.565 Then
                    sldDiffEstimate = 3
                ElseIf dblIRT < 1.197 Then
                    sldDiffEstimate = 2
                Else
                    sldDiffEstimate = 1
                End If
        End Select

    End Sub
```

```
' frmDrag.frm
VERSION 5.00
Begin VB.Form frmDrag
  Caption       =  "Window drag control"
  ClientHeight   =  1005
  ClientLeft     =  60
  ClientTop      =  345
  ClientWidth    =  3060
  LinkTopic      =  "Form1"
  ScaleHeight    =  1005
  ScaleWidth     =  3060
  StartUpPosition =  2  'CenterScreen
  Begin VB.CommandButton Command2
    Caption        =  "Full Drag OFF"
    Height         =  735
    Left           =  1560
    TabIndex       =  1
    Top            =  120
    Width          =  1215
  End
  Begin VB.CommandButton Command1
    Caption        =  "Full Drag ON"
    Height         =  735
    Left           =  120
    TabIndex       =  0
    Top            =  120
    Width          =  1215
  End
End
Attribute VB_Name = "frmDrag"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Declare Function SystemParametersInfo Lib "user32" _
   Alias "SystemParametersInfoA" (ByVal uAction As Long, _
   ByVal uParam As Long, ByRef lpvParam As Any, _
   ByVal fuWinIni As Long) As Long

Private Const SPI_GETDRAGFULLWINDOWS = 38
Private Const SPI_SETDRAGFULLWINDOWS = 37
Private Const SPIF_SENDWININICHANGE = 2
```

```
Public Function IsFullWindowDragOn() As Boolean

    Dim result As Long

    'Call API and check for successful call.
    If SystemParametersInfo(SPI_GETDRAGFULLWINDOWS, 0&, result, 0&) <> 0 Then
        'Feature supported now check value of result.
        If result = 0 Then
            IsFullWindowDragOn = False
        Else
            IsFullWindowDragOn = True
        End If
        'Call failed, feature not supported.
    Else
        IsFullWindowDragOn = False
    End If

End Function

Private Sub TurnOffFullWindowDrag()

    Dim result As Long

    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 0&, _
        ByVal vbNullString, SPIF_SENDWININICHANGE)

End Sub

Private Sub TurnOnFullWindowDrag()

    Dim result As Long

    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 1&, _
        ByVal vbNullString, SPIF_SENDWININICHANGE)

End Sub

Private Sub Command1_Click()

    TurnOnFullWindowDrag

End Sub

Private Sub Command2_Click()
```

```
    TurnOffFullWindowDrag

End Sub
```

```
' frmIED.frm
VERSION 5.00
Begin VB.Form frmIED
  BorderStyle    =  1  'Fixed Single
  Caption        =  "TCA Installation"
  ClientHeight   =  1185
  ClientLeft     =  45
  ClientTop      =  330
  ClientWidth    =  2475
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  1185
  ScaleWidth     =  2475
  StartUpPosition =  2  'CenterScreen
  Begin VB.CommandButton cmdOK
    Caption        =  "OK"
    Height         =  375
    Left           =  600
    TabIndex       =  1
    Top            =  720
    Width          =  1215
  End
  Begin VB.Label Label1
    Caption        =  "Setting IED files to read-only."
    Height         =  255
    Left           =  240
    TabIndex       =  0
    Top            =  240
    Width          =  2055
  End
End
Attribute VB_Name = "frmIED"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdOK_Click()

   Unload Me

End Sub
```

```
Private Sub Form_Load()

   Call Shell("attrib +r C:\tcs\working\dscbt.ied", vbHide)
   Call Shell("attrib +r C:\tcs\working\qccbt.ied", vbHide)
   Call Shell("attrib +r C:\tcs\working\qcppt.ied", vbHide)
   Call Shell("attrib +r C:\tcs\working\ssmccbt.ied", vbHide)
   Call Shell("attrib +r C:\tcs\working\ssmcppt.ied", vbHide)

End Sub
```

```
' frmIndexedString.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Begin VB.Form frmIndexedString
  BorderStyle    =  4  'Fixed ToolWindow
  ClientHeight   =  2265
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  5835
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  2265
  ScaleWidth     =  5835
  ShowInTaskbar  =  0  'False
  StartUpPosition =  1  'CenterOwner
  Begin ComctlLib.ListView lvwIndexed
    Height         =  1815
    Left           =  120
    TabIndex       =  6
    Top            =  120
    Width          =  4215
    _ExtentX       =  7435
    _ExtentY       =  3201
    View           =  3
    Arrange        =  2
    LabelEdit      =  1
    MultiSelect    =  -1  'True
    LabelWrap      =  -1  'True
    HideSelection  =  0  'False
    _Version       =  327682
    ForeColor      =  -2147483640
    BackColor      =  -2147483643
    BorderStyle    =  1
    Appearance     =  1
    NumItems       =  2
    BeginProperty ColumnHeader(1) {0713E8C7-850A-101B-AFC0-4210102A8DA7}
      Key            =  ""
      Object.Tag          =  ""
      Text           =  "Index"
      Object.Width        =  529
    EndProperty
    BeginProperty ColumnHeader(2) {0713E8C7-850A-101B-AFC0-4210102A8DA7}
```

```
      SubItemIndex   =  1
      Key         =  ""
      Object.Tag          =  ""
      Text        =  "Value"
      Object.Width        =  6174
    EndProperty
  End
  Begin VB.CommandButton cmdAdd
    Caption       =  "Add"
    Height        =  255
    Left          =  120
    TabIndex      =  5
    ToolTipText    =  "Click here to add a value to the end of the list."
    Top           =  1900
    Width         =  975
  End
  Begin VB.CommandButton cmdInsert
    Caption       =  "Insert"
    Height        =  255
    Left          =  1080
    TabIndex      =  4
    ToolTipText    =  "Click here to insert a value before the currently selected value."
    Top           =  1900
    Width         =  1095
  End
  Begin VB.CommandButton cmdEdit
    Caption       =  "Edit"
    Height        =  255
    Left          =  2160
    TabIndex      =  3
    ToolTipText    =  "Click here to edit the currently selected value."
    Top           =  1900
    Width         =  1095
  End
  Begin VB.CommandButton cmdRemove
    Caption       =  "Remove"
    Height        =  255
    Left          =  3240
    TabIndex      =  2
    ToolTipText    =  "Click here to remove the selected value."
    Top           =  1900
    Width         =  1095
  End
  Begin VB.CommandButton cmdStrOK
    Caption       =  "OK"
```

```
      Default       =  -1  'True
      Height        =  495
      Left          =  4440
      TabIndex      =  0
      ToolTipText   =  "Click here to save changes and return."
      Top           =  120
      Width         =  1215
    End
    Begin VB.CommandButton cmdStrCancel
      Caption       =  "Cancel"
      Height        =  495
      Left          =  4440
      TabIndex      =  1
      ToolTipText   =  "Click here to return without saving changes."
      Top           =  720
      Width         =  1215
    End
    Begin VB.Menu mnuIndexed
      Caption       =  "Indexed"
      Visible       =  0  'False
      Begin VB.Menu mnuIndexedAdd
        Caption       =  "Add"
      End
      Begin VB.Menu mnuIndexedInsert
        Caption       =  "Insert"
      End
      Begin VB.Menu mnuIndexedEdit
        Caption       =  "Edit"
      End
      Begin VB.Menu mnuIndexedRemove
        Caption       =  "Remove"
      End
    End
  End
  Attribute VB_Name = "frmIndexedString"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = True
  Attribute VB_Exposed = False
  Option Explicit

  Private mudtModel As Model
  Private mudtEF As EditFlags
  Private mstrVariableName As String
  Private mcolStrings As Collection
```

```
Private mblnOK As Boolean
Public Property Let Model(ByVal udtNewValue As Model)
   Set mudtModel = udtNewValue
End Property
Public Property Let AddEditFlag(ByVal udtNewValue As EditFlags)
   mudtEF = udtNewValue
End Property
Public Property Let SubStringCollection(ByVal colNewValue As Collection)
   Set mcolStrings = colNewValue
End Property
Private Sub cmdAdd_Click()
   Call mnuIndexedAdd_Click
End Sub
Private Sub cmdEdit_Click()
   Call mnuIndexedEdit_Click
End Sub
Private Sub cmdInsert_Click()
   Call mnuIndexedInsert_Click
End Sub
Private Sub cmdRemove_Click()
   Call mnuIndexedRemove_Click
End Sub
Private Sub Form_Load()
```

```
    Dim varS As Variant
    Dim lsiLI As ListItem

    Dim udtWAPI As New Win32API

    ' enable full row select
    Call udtWAPI.EnableListViewFullRowSelect(lvwIndexed)

    mblnOK = False

    frmIndexedString.Caption = "Editing substrings of string " & mstrVariableName

    If mudtEF = aeEdit Then
        With lvwIndexed
            For Each varS In mcolStrings
                Set lsiLI = .ListItems.Add
                UpdateListView
                lsiLI.SubItems(1) = varS
            Next varS
        End With
    End If

    ' prevent changes if model is frozen
    If mudtModel.IsFrozen Then
        cmdStrOK.Enabled = False
        cmdAdd.Enabled = False
        mnuIndexedAdd.Enabled = False
        cmdEdit.Caption = "Browse"
        mnuIndexedEdit.Caption = "Browse"
        cmdInsert.Enabled = False
        mnuIndexedInsert.Enabled = False
        cmdRemove.Enabled = False
        mnuIndexedRemove.Enabled = False
    End If

End Sub

Public Property Let VariableName(ByVal strNewValue As String)

    mstrVariableName = strNewValue

End Property

Public Property Get StringValue() As String
```

```
    Dim udtSS As New SubString

    udtSS.Delimiter = Chr(STRING_DELIMITER)
    udtSS.StringCollection = mcolStrings
    StringValue = udtSS.StringValue

End Property

Public Property Get SubStringCollection() As Collection

    Set SubStringCollection = mcolStrings

End Property

Public Property Get OK() As Boolean

    OK = mblnOK

End Property

Private Sub cmdStrOK_Click()

    Dim lsiItem As ListItem

    Set mcolStrings = New Collection

    For Each lsiItem In lvwIndexed.ListItems
        Call mcolStrings.Add(lsiItem.SubItems(1))
    Next lsiItem

    mblnOK = True

    Unload Me

End Sub

Private Sub cmdStrCancel_Click()

    Unload Me

End Sub

Private Sub mnuIndexedAdd_Click()

    With frmString
```

```
        ' set the model
        .Model = mudtModel
        ' set the string
        .StringValue = ""
        ' set var name
        .VariableName = mstrVariableName & "." _
           & Trim(Str(lvwIndexed.ListItems.Count + 1))
        ' do it
        .Show vbModal
        If .OK = False Then Exit Sub
    End With

    Dim lsiNewItem As ListItem

    Set lsiNewItem = lvwIndexed.ListItems.Add
    UpdateListView
    lsiNewItem.SubItems(1) = frmString.StringValue

End Sub

Private Sub mnuIndexedEdit_Click()

    With frmString
        ' set the model
        .Model = mudtModel
        ' set the string
        .StringValue = lvwIndexed.SelectedItem.SubItems(1)
        ' set var name
        .VariableName = mstrVariableName & "." _
           & Trim(Str(lvwIndexed.SelectedItem.Index))
        ' do it
        .Show vbModal
        If .OK = False Then Exit Sub
    End With

    lvwIndexed.SelectedItem.SubItems(1) = frmString.StringValue

End Sub

Private Sub mnuIndexedInsert_Click()

    If lvwIndexed.SelectedItem Is Nothing Then Exit Sub

    With frmString
        ' set the Model
        .Model = mudtModel
        ' set the string
        .StringValue = ""
        ' set var name
        .VariableName = mstrVariableName
        ' do it
        .Show vbModal
        If .OK = False Then Exit Sub
    End With

    Dim lsiNewItem As ListItem

    Set lsiNewItem = lvwIndexed.ListItems.Add(lvwIndexed.SelectedItem.Index)
    UpdateListView
    lsiNewItem.SubItems(1) = frmString.StringValue

End Sub

Private Sub mnuIndexedRemove_Click()

    If lvwIndexed.SelectedItem Is Nothing Then Exit Sub

    Call lvwIndexed.ListItems.Remove(lvwIndexed.SelectedItem.Index)
    UpdateListView

End Sub

Private Sub UpdateListView()

    Dim intI As Integer

    For intI = 1 To lvwIndexed.ListItems.Count
        lvwIndexed.ListItems.Item(intI).Text = Str(intI)
    Next intI

End Sub
```

```
' frmNew.frm
VERSION 5.00
Begin VB.Form frmNew
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "New family properties"
  ClientHeight   =  1740
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  6240
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  1740
  ScaleWidth     =  6240
  ShowInTaskbar  =  0   'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.CommandButton cmdCancel
    Cancel         =  -1  'True
    Caption        =  "Cancel"
    Height         =  495
    Left           =  4800
    TabIndex       =  9
    Top            =  720
    Width          =  1215
  End
  Begin VB.CommandButton cmdOK
    Caption        =  "OK"
    Default        =  -1  'True
    Height         =  495
    Left           =  4800
    TabIndex       =  8
    Top            =  120
    Width          =  1215
  End
  Begin VB.OptionButton optGeneric
    Caption        =  "Non-generic"
    Height         =  195
    Index          =  1
    Left           =  3240
    TabIndex       =  7
    Top            =  1150
    Width          =  1455
  End
```

```
Begin VB.OptionButton optGeneric
  Caption       =  "Generic"
  Height        =  195
  Index         =  0
  Left          =  2280
  TabIndex      =  6
  Top           =  1150
  Value         =  -1  'True
  Width         =  975
End
Begin VB.ComboBox cboProximity
  Height        =  315
  ItemData      =  "frmNew.frx":0000
  Left          =  2280
  List          =  "frmNew.frx":000D
  Style         =  2  'Dropdown List
  TabIndex      =  4
  Top           =  360
  Width         =  1935
End
Begin VB.ComboBox cboItemType
  Height        =  315
  ItemData      =  "frmNew.frx":0024
  Left          =  120
  List          =  "frmNew.frx":0031
  Style         =  2  'Dropdown List
  TabIndex      =  2
  Top           =  1080
  Width         =  1935
End
Begin VB.ComboBox cboProgram
  Height        =  315
  ItemData      =  "frmNew.frx":0072
  Left          =  120
  List          =  "frmNew.frx":007F
  Style         =  2  'Dropdown List
  TabIndex      =  0
  Top           =  360
  Width         =  1935
End
Begin VB.Label lbl
  Caption       =  "Variant proximity"
  Height        =  255
  Left          =  2280
  TabIndex      =  5
```

```
      Top          =  120
      Width         =  1335
    End
    Begin VB.Label lblItemType
      Caption        =  "Item type"
      Height        =  255
      Left         =  120
      TabIndex       =  3
      Top          =  840
      Width         =  1335
    End
    Begin VB.Label lblProgram
      Caption        =  "Program"
      Height        =  255
      Left         =  120
      TabIndex       =  1
      Top          =  120
      Width         =  1335
    End
End
Attribute VB_Name = "frmNew"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mblnOK As Boolean

Private mudtProgram As Program
Private mudtItemType As ItemType
Private mudtProximity As Proximity
Private mblnGeneric As Boolean

Private Sub Form_Load()

   mblnOK = False

   ' init combo boxes
   cboProgram.ListIndex = 0
   cboItemType.ListIndex = 0
   cboProximity.ListIndex = 0

End Sub
```

```
Public Property Get OK() As Boolean
   OK = mblnOK
End Property
Public Property Get Program() As Program
   Program = mudtProgram
End Property
Public Property Get ItemType() As ItemType
   ItemType = mudtItemType
End Property
Public Property Get Proximity() As Proximity
   Proximity = mudtProximity
End Property
Public Property Get Generic() As Boolean
   Generic = mblnGeneric
End Property
Private Sub cboProgram_Click()
   mudtProgram = cboProgram.ListIndex
End Sub
Private Sub cboItemType_Click()
   mudtItemType = cboItemType.ListIndex
End Sub
Private Sub cboProximity_Click()
   mudtProximity = cboProximity.ListIndex
```

```
End Sub
Private Sub optGeneric_Click(Index As Integer)
   mblnGeneric = optGeneric(0)
End Sub
Private Sub cmdOK_Click()
   mblnOK = True
   Unload Me
End Sub
Private Sub cmdCancel_Click()
   Unload Me
End Sub
```

```
' frmNewModel.frm
VERSION 5.00
Begin VB.Form frmNewFamily
  BorderStyle    =  4 'Fixed ToolWindow
  Caption        =  "New family"
  ClientHeight   =  1350
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  4680
  LinkTopic      =  "Form1"
  LockControls   =  -1 'True
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  1350
  ScaleWidth     =  4680
  ShowInTaskbar  =  0  'False
  StartUpPosition =  1 'CenterOwner
  Begin VB.OptionButton optModelType
    Caption        =  "Quantitative Comparision"
    Height         =  255
    Index          =  1
    Left           =  480
    TabIndex       =  4
    Top            =  480
    Width          =  2535
  End
  Begin VB.OptionButton optModelType
    Caption        =  "Data Sufficiency"
    Height         =  255
    Index          =  2
    Left           =  480
    TabIndex       =  3
    Top            =  720
    Width          =  2535
  End
  Begin VB.OptionButton optModelType
    Caption        =  "Standard Multiple Choice"
    Height         =  255
    Index          =  0
    Left           =  480
    TabIndex       =  2
    Top            =  240
    Value          =  -1 'True
    Width          =  2535
```

```
      End
      Begin VB.CommandButton cmdCancel
        Caption       =  "Cancel"
        Height        =  495
        Left        =  3360
        TabIndex      =  1
        ToolTipText    =  "Click here to return without opening creating a new model."
        Top          =  720
        Width        =  1215
      End
      Begin VB.CommandButton cmdNewCreate
        Caption       =  "Create"
        Default       =  -1  'True
        Height        =  495
        Left        =  3360
        TabIndex      =  0
        ToolTipText    =  "Click here to create the new family."
        Top          =  120
        Width        =  1215
      End
    End
    Attribute VB_Name = "frmNewFamily"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = True
    Attribute VB_Exposed = False
    Option Explicit

    Private mblnOK As Boolean

    ' holds the item type
    Private mudtItemType As ItemType

    Public Property Get OK() As Boolean

       OK = mblnOK

    End Property

    Public Property Get ItemType() As ItemType

       ItemType = mudtItemType

    End Property
```

```
Private Sub cmdNewCreate_Click()
   mblnOK = True
   Unload Me
End Sub
Private Sub cmdCancel_Click()
   mblnOK = False
   Unload Me
End Sub
Private Sub optModelType_Click(Index As Integer)
   mudtItemType = Index
End Sub
```

```
' frmProgram.frm
VERSION 5.00
Begin VB.Form frmProgram
  Caption       =  "Select the program"
  ClientHeight   =  1350
  ClientLeft     =  60
  ClientTop      =  345
  ClientWidth    =  3225
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  ScaleHeight    =  1350
  ScaleWidth     =  3225
  StartUpPosition =  1  'CenterOwner
  Begin VB.OptionButton optProgram
    Caption       =  "SAT"
    Height        =  195
    Index         =  2
    Left          =  240
    TabIndex      =  4
    Top           =  720
    Width         =  1335
  End
  Begin VB.OptionButton optProgram
    Caption       =  "GMAT"
    Height        =  195
    Index         =  1
    Left          =  240
    TabIndex      =  3
    Top           =  480
    Width         =  1335
  End
  Begin VB.OptionButton optProgram
    Caption       =  "GRE"
    Height        =  195
    Index         =  0
    Left          =  240
    TabIndex      =  2
    Top           =  240
    Value         =  -1  'True
    Width         =  1335
  End
  Begin VB.CommandButton cmdCancel
    Caption       =  "Cancel"
    Height        =  495
```

```
      Left           =  1920
      TabIndex       =  1
      ToolTipText    =  "Click here to return."
      Top            =  720
      Width          =  1215
   End
   Begin VB.CommandButton cmdOK
      Caption        =  "OK"
      Height         =  495
      Left           =  1920
      TabIndex       =  0
      ToolTipText    =  "Click here to save the currently selected program and return."
      Top            =  120
      Width          =  1215
   End
End
Attribute VB_Name = "frmProgram"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mblnOK As Boolean

Private mudtProgram As Program

Public Property Get OK() As Boolean

    OK = mblnOK

End Property

Public Property Get Program() As Program

    Program = mudtProgram

End Property

Private Sub cmdOK_Click()

    mblnOK = True

    Unload Me
```

```
End Sub
Private Sub cmdCancel_Click()
   mblnOK = False
   Unload Me
End Sub
Private Sub optProgram_Click(Index As Integer)
   mudtProgram = Index
End Sub
```

```
' frmProgress.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.2#0"; "COMCTL32.OCX"
Begin VB.Form frmProgress
  BorderStyle    =  1  'Fixed Single
  ClientHeight   =  1110
  ClientLeft     =  15
  ClientTop      =  15
  ClientWidth    =  4500
  ClipControls   =  0  'False
  ControlBox     =  0  'False
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  1110
  ScaleWidth     =  4500
  StartUpPosition =  2  'CenterScreen
  Begin ComctlLib.ProgressBar prbProgressBar
    Height         =  255
    Left           =  240
    TabIndex       =  0
    Top            =  600
    Width          =  3975
    _ExtentX       =  7011
    _ExtentY       =  450
    _Version       =  327682
    Appearance     =  1
    Max            =  500
  End
  Begin VB.Label lblProgress
    Alignment      =  2  'Center
    BeginProperty Font
      Name           =  "MS Sans Serif"
      Size           =  8.25
      Charset        =  0
      Weight         =  700
      Underline      =  0  'False
      Italic         =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height         =  255
    Left           =  240
    TabIndex       =  1
    Top          =  240
    Width         =  3855
  End
End
Attribute VB_Name = "frmProgress"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
```

```
' frmProlog.frm
VERSION 5.00
Begin VB.Form frmProlog
  BorderStyle    =  5  'Sizable ToolWindow
  ClientHeight   =  900
  ClientLeft     =  2775
  ClientTop      =  3720
  ClientWidth    =  4440
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  900
  ScaleWidth     =  4440
  ShowInTaskbar  =  0   'False
  StartUpPosition =  2  'CenterScreen
  Begin VB.CommandButton cmdAbort
    Caption        =  "Abort"
    Default        =  -1  'True
    Height         =  495
    Left           =  3120
    TabIndex       =  0
    Top            =  120
    Width          =  1215
  End
  Begin VB.Label lblProlog
    Height         =  495
    Left           =  120
    TabIndex       =  1
    Top            =  120
    Width          =  2655
  End
End
Attribute VB_Name = "frmProlog"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Option Explicit

Private mblnAbort As Boolean

Public Property Get Abort() As Boolean
```

```
    Abort = mblnAbort

End Property

Public Sub Kill()

    Unload Me

End Sub

Private Sub Form_Load()

    mblnAbort = False

End Sub

Private Sub cmdAbort_Click()

    mblnAbort = True
    Unload Me

End Sub
```

```
' frmSplash.frm
VERSION 5.00
Begin VB.Form frmSplash
  BorderStyle     =  3  'Fixed Dialog
  ClientHeight    =  4245
  ClientLeft      =  255
  ClientTop       =  1410
  ClientWidth     =  7380
  ClipControls    =  0   'False
  ControlBox      =  0   'False
  Icon            =  "frmSplash.frx":0000
  KeyPreview      =  -1  'True
  LinkTopic       =  "Form2"
  LockControls    =  -1  'True
  MaxButton       =  0   'False
  MinButton       =  0   'False
  ScaleHeight     =  4245
  ScaleWidth      =  7380
  ShowInTaskbar   =  0   'False
  StartUpPosition =  2  'CenterScreen
  Begin VB.Frame fraSplash
    Height        =  4050
    Left          =  120
    TabIndex      =  0
    Top           =  60
    Width         =  7080
    Begin VB.Image imgLogo
      BorderStyle    =  1  'Fixed Single
      Height         =  780
      Left           =  600
      Picture        =  "frmSplash.frx":000C
      Top            =  720
      Width          =  1275
    End
    Begin VB.Label lblCopyright
      Caption        =  "Copyright 1999"
      BeginProperty Font
        Name           =  "Arial"
        Size           =  8.25
        Charset        =  0
        Weight         =  400
        Underline      =  0  'False
        Italic         =  0  'False
        Strikethrough  =  0  'False
    EndProperty
    Height        =  255
    Left        =  4560
    TabIndex      =  3
    Top         =  3480
    Width        =  2415
  End
  Begin VB.Label lblCompany
    Caption       =  "Educational Testing Service"
    BeginProperty Font
      Name         =  "Arial"
      Size         =  8.25
      Charset       =  0
      Weight        =  400
      Underline      =  0  'False
      Italic        =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height        =  255
    Left        =  4560
    TabIndex      =  2
    Top         =  3720
    Width        =  2415
  End
  Begin VB.Label lblWarning
    Caption       =  "Proprietary and Confidential"
    BeginProperty Font
      Name         =  "Arial"
      Size         =  9.75
      Charset       =  0
      Weight        =  700
      Underline      =  0  'False
      Italic        =  0  'False
      Strikethrough  =  0  'False
    EndProperty
    Height        =  315
    Left        =  240
    TabIndex      =  1
    Top         =  3600
    Width        =  2775
  End
  Begin VB.Label lblVersion
    Alignment      =  1  'Right Justify
    AutoSize       =  -1  'True
    Caption       =  "Version 1.25"
    BeginProperty Font
      Name          =   "Arial"
      Size          =   12
      Charset       =   0
      Weight        =   700
      Underline     =   0   'False
      Italic        =   0   'False
      Strikethrough =   0   'False
    EndProperty
    Height          =   285
    Left            =   5265
    TabIndex        =   4
    Top             =   2880
    Width           =   1410
  End
  Begin VB.Label lblProductName
    AutoSize        =   -1  'True
    Caption         =   "Assistant"
    BeginProperty Font
      Name          =   "Arial"
      Size          =   48
      Charset       =   0
      Weight        =   700
      Underline     =   0   'False
      Italic        =   0   'False
      Strikethrough =   0   'False
    EndProperty
    Height          =   1125
    Left            =   1440
    TabIndex        =   6
    Top             =   1560
    Width           =   4320
  End
  Begin VB.Label lblCompanyProduct
    AutoSize        =   -1  'True
    Caption         =   "Test Creation "
    BeginProperty Font
      Name          =   "Arial"
      Size          =   18
      Charset       =   0
      Weight        =   700
      Underline     =   0   'False
      Italic        =   0   'False
      Strikethrough =   0   'False
    EndProperty
```

```
        Height      =  435
        Left       =  2400
        TabIndex     =  5
        Top        =  1080
        Width       =  2400
      End
    End
End
Attribute VB_Name = "frmSplash"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Option Explicit

Public Sub UnloadMe()

    Unload Me

End Sub
```

```
' SetPrecision.frm
VERSION 5.00
Begin VB.Form frmSetPrecision
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "Set Precision"
  ClientHeight   =  1965
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  3540
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  1965
  ScaleWidth     =  3540
  ShowInTaskbar  =  0   'False
  StartUpPosition =  2  'CenterScreen
  Begin VB.CommandButton cmdSetPrecisionDefault
    Caption       =  "Default"
    Height        =  495
    Left          =  2160
    TabIndex      =  3
    ToolTipText   =  "Click here to return to the default value for precision."
    Top           =  1320
    Width         =  1215
  End
  Begin VB.CommandButton cmdSetPrecisionOK
    Caption       =  "OK"
    Default       =  -1  'True
    Height        =  495
    Left          =  2160
    TabIndex      =  2
    ToolTipText   =  "Click here to save the displayed value."
    Top           =  120
    Width         =  1215
  End
  Begin VB.CommandButton cmdSetPrecisionCancel
    Caption       =  "Cancel"
    Height        =  495
    Left          =  2160
    TabIndex      =  1
    ToolTipText   =  "Click here to return without saving any changes to precision."
    Top           =  720
    Width         =  1215
  End
  Begin VB.TextBox txtPrecision
    Height       =  315
    Left         =  120
    TabIndex     =  0
    Text         =  ".1"
    Top          =  120
    Width        =  1815
  End
End
Attribute VB_Name = "frmSetPrecision"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdSetPrecisionCancel_Click()

   Unload Me

End Sub

Private Sub cmdSetPrecisionDefault_Click()

   txtPrecision = ".001"

End Sub

Private Sub cmdSetPrecisionOK_Click()

   frmTCA.Precision = txtPrecision
   Unload Me

End Sub

Private Sub Form_Load()

   txtPrecision = frmTCA.Precision

End Sub

Private Sub txtPrecision_GotFocus()

   ' Automatically select all text when TextBox gets focus
   Call txtSelectAll(txtPrecision)
```

```
End Sub
```

```
' String.frm
VERSION 5.00
Begin VB.Form frmString
  BorderStyle    =  4  'Fixed ToolWindow
  ClientHeight   =  2265
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  5835
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  2265
  ScaleWidth     =  5835
  ShowInTaskbar  =  0   'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.CommandButton cmdStrOK
    Caption        =  "OK"
    Default        =  -1  'True
    Height         =  495
    Left           =  4440
    TabIndex       =  1
    ToolTipText    =  "Click here to save changes and return."
    Top            =  120
    Width          =  1215
  End
  Begin VB.CommandButton cmdStrCancel
    Caption        =  "Cancel"
    Height         =  495
    Left           =  4440
    TabIndex       =  2
    ToolTipText    =  "Click here to return without saving changes."
    Top            =  720
    Width          =  1215
  End
  Begin VB.TextBox txtString
    Height         =  315
    Left           =  240
    TabIndex       =  0
    Top            =  480
    Width          =  3975
  End
End
Attribute VB_Name = "frmString"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mudtModel As Model
Private mstrVariableName As String
Private mstrStringValue As String
Private mblnOK As Boolean

Public Property Let Model(ByVal udtNewValue As Model)

   Set mudtModel = udtNewValue

End Property

Public Property Let VariableName(ByVal strNewValue As String)

   mstrVariableName = strNewValue

End Property

Public Property Let StringValue(ByVal strNewValue As String)

   mstrStringValue = strNewValue

End Property

Public Property Get StringValue() As String

   StringValue = mstrStringValue

End Property

Public Property Get OK() As Boolean

   OK = mblnOK

End Property

Private Sub Form_Load()

   mblnOK = False
```

```
    frmString.Caption = "Editing string " & mstrVariableName

    txtString = mstrStringValue

    If mudtModel.IsFrozen Then
      cmdStrOK.Enabled = False
    End If

End Sub

Private Sub cmdStrOK_Click()

    mblnOK = True
    StringValue = txtString

    Unload Me

End Sub

Private Sub cmdStrCancel_Click()

    Unload Me

End Sub

Private Sub txtString_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtString)

End Sub
```

```
' TCA.FRM
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Object = "{BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0"; "TABCTL32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
Begin VB.Form frmTCA
  Caption        =  "ETS Test Creation Assistant"
  ClientHeight   =  8310
  ClientLeft     =  165
  ClientTop      =  735
  ClientWidth    =  11400
  LinkTopic      =  "Form1"
  LockControls   =  -1  'True
  ScaleHeight    =  8310
  ScaleWidth     =  11400
  StartUpPosition =  3  'Windows Default
  Begin VB.Frame frmDummy
    Caption        =  "Common dialog anchor"
    Height         =  855
    Left           =  2640
    TabIndex       =  3
    Top            =  2280
    Visible        =  0  'False
    Width          =  2055
    Begin MSComDlg.CommonDialog cdlCD
      Left           =  120
      Top            =  240
      _ExtentX       =  847
      _ExtentY       =  847
      _Version       =  393216
    End
  End
  Begin VB.Frame fraWord
    Height         =  8535
    Left           =  120
    TabIndex       =  1
    Top            =  0
    Width          =  6255
  End
  Begin TabDlg.SSTab sstMainTab
    Height         =  8535
    Left           =  6480
    TabIndex       =  0
    Top            =  0
```

```
Width          =  5655
_ExtentX       =  9975
_ExtentY       =  15055
_Version       =  393216
TabHeight      =  520
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
  Name           =  "MS Sans Serif"
  Size           =  8.25
  Charset        =  0
  Weight         =  400
  Underline      =  0  'False
  Italic         =  0  'False
  Strikethrough  =  0  'False
EndProperty
TabCaption(0)  =  "Family Overview"
TabPicture(0)  =  "TCA.frx":0000
Tab(0).ControlEnabled=  -1  'True
Tab(0).Control(0)=  "lblFamily"
Tab(0).Control(0).Enabled=  0  'False
Tab(0).Control(1)=  "imlI"
Tab(0).Control(1).Enabled=  0  'False
Tab(0).Control(2)=  "lblDummy"
Tab(0).Control(2).Enabled=  0  'False
Tab(0).Control(3)=  "lblAccepted"
Tab(0).Control(3).Enabled=  0  'False
Tab(0).Control(4)=  "lstAccepted"
Tab(0).Control(4).Enabled=  0  'False
Tab(0).Control(5)=  "txtVariablize"
Tab(0).Control(5).Enabled=  0  'False
Tab(0).Control(6)=  "treModels"
Tab(0).Control(6).Enabled=  0  'False
Tab(0).Control(7)=  "cmdSetAttributes"
Tab(0).Control(7).Enabled=  0  'False
Tab(0).Control(8)=  "lstDummy"
Tab(0).Control(8).Enabled=  0  'False
Tab(0).Control(9)=  "cmdDone"
Tab(0).Control(9).Enabled=  0  'False
Tab(0).Control(10)=  "cmdPrintBatch"
Tab(0).Control(10).Enabled=  0  'False
Tab(0).Control(11)=  "cmdTreeExtend"
Tab(0).Control(11).Enabled=  0  'False
Tab(0).Control(12)=  "cmdTreeRemove"
Tab(0).Control(12).Enabled=  0  'False
Tab(0).Control(13)=  "cmdAcceptedPaste"
Tab(0).Control(13).Enabled=  0  'False
Tab(0).Control(14)=  "cmdAcceptedCopy"
Tab(0).Control(14).Enabled=  0  'False
Tab(0).Control(15)=  "cmdAcceptedEdit"
Tab(0).Control(15).Enabled=  0  'False
Tab(0).ControlCount=  16
TabCaption(1)  =  "Model Workshop"
TabPicture(1)  =  "TCA.frx":001C
Tab(1).ControlEnabled=  0  'False
Tab(1).Control(0)=  "lblVariables"
Tab(1).Control(1)=  "lblCloningConstraints"
Tab(1).Control(2)=  "lblDistractor"
Tab(1).Control(3)=  "cmdExportConstraints"
Tab(1).Control(4)=  "cmdImportConstraints"
Tab(1).Control(5)=  "cmdSaveModel"
Tab(1).Control(6)=  "cmdTestAll"
Tab(1).Control(7)=  "lstConstraints(1)"
Tab(1).Control(8)=  "cmdVariableAdd"
Tab(1).Control(9)=  "cmdVariableEdit"
Tab(1).Control(10)=  "cmdVariableRemove"
Tab(1).Control(11)=  "cmdVariableTest"
Tab(1).Control(12)=  "cmdConstraintAdd(0)"
Tab(1).Control(13)=  "cmdConstraintEdit(0)"
Tab(1).Control(14)=  "cmdConstraintRemove(0)"
Tab(1).Control(15)=  "cmdConstraintTest(0)"
Tab(1).Control(16)=  "cmdConstraintAdd(1)"
Tab(1).Control(17)=  "cmdConstraintEdit(1)"
Tab(1).Control(18)=  "cmdConstraintRemove(1)"
Tab(1).Control(19)=  "cmdConstraintTest(1)"
Tab(1).Control(20)=  "cmdPrintConstraints"
Tab(1).Control(21)=  "lstConstraints(0)"
Tab(1).Control(22)=  "lstVariables"
Tab(1).Control(23)=  "cmdComments"
Tab(1).ControlCount=  24
TabCaption(2)  =  "Generate Variants"
TabPicture(2)  =  "TCA.frx":0038
Tab(2).ControlEnabled=  0  'False
Tab(2).Control(0)=  "cmdDispMakeModel"
Tab(2).Control(1)=  "cmdDispDiscard"
Tab(2).Control(2)=  "cmdDispDefer"
Tab(2).Control(3)=  "cmdDispAccept"
Tab(2).Control(4)=  "sldDifference"
Tab(2).Control(5)=  "lstDisposition"
Tab(2).Control(6)=  "cmdPrintVariants"
Tab(2).Control(7)=  "cmdDisplayModel"
Tab(2).Control(8)=  "txtNum2Generate"
Tab(2).Control(9)=  "cmdGenerate"
Tab(2).Control(10)=  "lblDiff"
Tab(2).Control(11)=  "Label1"
Tab(2).Control(12)=  "lblMed"
Tab(2).Control(13)=  "lblLow"
Tab(2).Control(14)=  "lblVariants"
Tab(2).Control(15)=  "LblNumVariants"
Tab(2).ControlCount=  16
Begin VB.CommandButton cmdComments
  Caption        =  "Comments"
  Height         =  495
  Left           =  -70680
  TabIndex       =  58
  ToolTipText    =  "Click here to print all variables and constraints."
  Top            =  3720
  Width          =  1215
End
Begin VB.ListBox lstVariables
  DragIcon       =  "TCA.frx":0054
  Height         =  1635
  ItemData       =  "TCA.frx":035E
  Left           =  -74760
  List           =  "TCA.frx":0360
  Style          =  1  'Checkbox
  TabIndex       =  57
  ToolTipText    =  "Left button click to select a constraint.  Then right button click for
constraint options."
  Top            =  720
  Width          =  3855
End
Begin VB.ListBox lstConstraints
  DragIcon       =  "TCA.frx":0362
  Height         =  1635
  Index          =  0
  ItemData       =  "TCA.frx":066C
  Left           =  -74760
  List           =  "TCA.frx":066E
  Style          =  1  'Checkbox
  TabIndex       =  56
  ToolTipText    =  "Left button click to select a constraint.  Then right button click for
constraint options."
  Top            =  3120
  Width          =  3855
End
Begin VB.CommandButton cmdAcceptedEdit
```

```
    Caption         =   "Edit Profile"
    Height          =   255
    Left            =   240
    TabIndex        =   54
    ToolTipText     =   "Click here to edit the profile of the selected variant."
    Top             =   7300
    Width           =   1335
  End
  Begin VB.CommandButton cmdAcceptedCopy
    Caption         =   "Copy Profile"
    Height          =   255
    Left            =   1560
    TabIndex        =   53
    ToolTipText     =   "Click here to copy the profile of the selected variant."
    Top             =   7300
    Width           =   1335
  End
  Begin VB.CommandButton cmdAcceptedPaste
    Caption         =   "Paste Profile"
    Height          =   255
    Left            =   2880
    TabIndex        =   52
    ToolTipText     =   "Click here to paste a profile onto the currently selected variants."
    Top             =   7300
    Width           =   1215
  End
  Begin VB.CommandButton cmdPrintConstraints
    Caption         =   "Print Constraints"
    Height          =   495
    Left            =   -70680
    TabIndex        =   51
    ToolTipText     =   "Click here to print all variables and constraints."
    Top             =   3120
    Width           =   1215
  End
  Begin VB.CommandButton cmdDispMakeModel
    Caption         =   "Create Mdl."
    Height          =   255
    Left            =   -71880
    TabIndex        =   50
    ToolTipText     =   "Click here to create new children of the active model using the
currently selected variants."
    Top             =   6120
    Width           =   975
  End
```

```
Begin VB.CommandButton cmdDispDiscard
  Caption         =   "Discard"
  Height          =   255
  Left            =   -72840
  TabIndex        =   49
  ToolTipText     =   "Click here to discard the currently selected variants."
  Top             =   6120
  Width           =   975
End
Begin VB.CommandButton cmdDispDefer
  Caption         =   "Defer"
  Height          =   255
  Left            =   -73800
  TabIndex        =   48
  ToolTipText     =   "Click here to defer the currently selected variants."
  Top             =   6120
  Width           =   975
End
Begin VB.CommandButton cmdDispAccept
  Caption         =   "Accept"
  Height          =   255
  Left            =   -74760
  TabIndex        =   47
  ToolTipText     =   "Click here to accept the currently selected variants."
  Top             =   6120
  Width           =   975
End
Begin VB.CommandButton cmdTreeRemove
  Caption         =   "Remove"
  Height          =   255
  Left            =   2160
  TabIndex        =   46
  ToolTipText     =   "Click here to remove a model."
  Top             =   3720
  Width           =   1935
End
Begin VB.CommandButton cmdTreeExtend
  Caption         =   "Extend"
  Height          =   255
  Left            =   240
  TabIndex        =   45
  ToolTipText     =   "Click here to create a new child of the selected model."
  Top             =   3720
  Width           =   1935
End
```

```
Begin VB.CommandButton cmdConstraintTest
  Caption       =  "Test"
  Height        =  255
  Index         =  1
  Left          =  -71880
  TabIndex      =  44
  ToolTipText   =  "Click here to test all enabled variables and distractor constraints."
  Top           =  7200
  Width         =  975
End
Begin VB.CommandButton cmdConstraintRemove
  Caption       =  "Remove"
  Height        =  255
  Index         =  1
  Left          =  -72840
  TabIndex      =  43
  ToolTipText   =  "Click here to remove a distractor constraint."
  Top           =  7200
  Width         =  975
End
Begin VB.CommandButton cmdConstraintEdit
  Caption       =  "Edit"
  Height        =  255
  Index         =  1
  Left          =  -73800
  TabIndex      =  42
  ToolTipText   =  "Click here to edit the currently selected distractor constraint."
  Top           =  7200
  Width         =  975
End
Begin VB.CommandButton cmdConstraintAdd
  Caption       =  "Add"
  Height        =  255
  Index         =  1
  Left          =  -74760
  TabIndex      =  41
  ToolTipText   =  "Click here to add a distractor constraint."
  Top           =  7200
  Width         =  975
End
Begin VB.CommandButton cmdConstraintTest
  Caption       =  "Test"
  Height        =  255
  Index         =  0
  Left          =  -71880
  TabIndex       =  40
  ToolTipText    =  "Click here to test all enabled variables and variation constraints."
  Top            =  4800
  Width          =  975
End
Begin VB.CommandButton cmdConstraintRemove
  Caption        =  "Remove"
  Height         =  255
  Index          =  0
  Left           =  -72840
  TabIndex       =  39
  ToolTipText    =  "Click here to remove the currently selected variation constraint."
  Top            =  4800
  Width          =  975
End
Begin VB.CommandButton cmdConstraintEdit
  Caption        =  "Edit"
  Height         =  255
  Index          =  0
  Left           =  -73800
  TabIndex       =  38
  ToolTipText    =  "Click here to edit the currently selected variation constraint."
  Top            =  4800
  Width          =  975
End
Begin VB.CommandButton cmdConstraintAdd
  Caption        =  "Add"
  Height         =  255
  Index          =  0
  Left           =  -74760
  TabIndex       =  37
  ToolTipText    =  "Click here to add a variation constraint."
  Top            =  4800
  Width          =  975
End
Begin VB.CommandButton cmdVariableTest
  Caption        =  "Test"
  Height         =  255
  Left           =  -71880
  TabIndex       =  36
  ToolTipText    =  "Click here to test all enabled variables."
  Top            =  2400
  Width          =  975
End
Begin VB.CommandButton cmdVariableRemove
```

```
        Caption       =  "Remove"
        Height        =  255
        Left          =  -72840
        TabIndex      =  35
        ToolTipText   =  "Click here to remove the currently selected variable."
        Top           =  2400
        Width         =  975
      End
      Begin VB.CommandButton cmdVariableEdit
        Caption       =  "Edit"
        Height        =  255
        Left          =  -73800
        TabIndex      =  34
        ToolTipText   =  "Click here to edit the currently selected variable."
        Top           =  2400
        Width         =  975
      End
      Begin VB.CommandButton cmdVariableAdd
        Caption       =  "Add"
        Height        =  255
        Left          =  -74760
        TabIndex      =  33
        ToolTipText   =  "Click here to add a variable."
        Top           =  2400
        Width         =  975
      End
      Begin VB.CommandButton cmdPrintBatch
        Caption       =  "Print All"
        Height        =  495
        Left          =  4320
        TabIndex      =  31
        ToolTipText   =  "Click here to print all variants."
        Top           =  4200
        Width         =  1215
      End
      Begin VB.CommandButton cmdDone
        Caption       =  "Done"
        Height        =  495
        Left          =  4320
        TabIndex      =  29
        ToolTipText   =  "Click here when you are done with this family and are ready to send it
   back to TCS."
        Top           =  1320
        Width         =  1215
      End
```

```
Begin ComctlLib.Slider sldDifference
  Height        =  255
  Left          =  -73440
  TabIndex      =  24
  ToolTipText   =  "Select the degree of randomization desired."
  Top           =  1140
  Width         =  1935
  _ExtentX      =  3413
  _ExtentY      =  450
  _Version      =  327682
  Max           =  2
  SelStart      =  2
  Value         =  2
End
Begin VB.ListBox lstDisposition
  Height        =  3570
  ItemData      =  "TCA.frx":0670
  Left          =  -74760
  List          =  "TCA.frx":0672
  MultiSelect   =  2  'Extended
  TabIndex      =  21
  ToolTipText   =  "Left button click to select a variant.  Then right button click for variant
options."
  Top           =  2520
  Width         =  3855
End
Begin VB.CommandButton cmdPrintVariants
  Caption       =  "Print All"
  Height        =  495
  Left          =  -70680
  TabIndex      =  20
  ToolTipText   =  "Click here to print all variants."
  Top           =  2400
  Width         =  1215
End
Begin VB.CommandButton cmdDisplayModel
  Caption       =  "Display Model"
  Height        =  495
  Left          =  -70680
  TabIndex      =  19
  ToolTipText   =  "Click here to view the active model."
  Top           =  1320
  Width         =  1215
End
Begin VB.ListBox lstDummy
```

```
      Height        =  255
      ItemData      =  "TCA.frx":0674
      Left          =  4680
      List          =  "TCA.frx":0676
      Sorted        =  -1  'True
      TabIndex      =  18
      Top           =  7800
      Visible       =  0   'False
      Width         =  615
   End
   Begin VB.TextBox txtNum2Generate
      Height        =  315
      Left          =  -74760
      TabIndex      =  16
      ToolTipText   =  "Enter the number variants to generate here."
      Top           =  1140
      Width         =  855
   End
   Begin VB.CommandButton cmdSetAttributes
      Caption       =  "Set Attributes"
      Enabled       =  0   'False
      Height        =  495
      Left          =  4320
      TabIndex      =  15
      ToolTipText   =  "Click here to reset the attributes for this model family."
      Top           =  720
      Width         =  1215
   End
   Begin ComctlLib.TreeView treModels
      DragIcon      =  "TCA.frx":0678
      Height        =  2955
      Left          =  240
      TabIndex      =  13
      ToolTipText   =  "Left button click on a model to select it.  Then right button click for
options."
      Top           =  780
      Width         =  3855
      _ExtentX      =  6800
      _ExtentY      =  5212
      _Version      =  327682
      LabelEdit     =  1
      LineStyle     =  1
      Style         =  7
      Appearance    =  1
   End
```

```
      Begin VB.ListBox lstConstraints
         DragIcon        =   "TCA.frx":07C2
         Height          =   1635
         Index           =   1
         ItemData        =   "TCA.frx":0ACC
         Left            =   -74760
         List            =   "TCA.frx":0ACE
         Style           =   1  'Checkbox
         TabIndex        =   10
         ToolTipText     =   "Left button click to select a constraint.  Then right button click for
   constraint options."
         Top             =   5520
         Width           =   3855
      End
      Begin VB.CommandButton cmdTestAll
         Caption         =   "Test All"
         Height          =   495
         Left            =   -70680
         TabIndex        =   8
         ToolTipText     =   "Click here to test all checked variables and constraints."
         Top             =   1320
         Width           =   1215
      End
      Begin VB.CommandButton cmdSaveModel
         Caption         =   "Save Model"
         Height          =   495
         Left            =   -70680
         TabIndex        =   7
         ToolTipText     =   "Click here to save this model."
         Top             =   720
         Width           =   1215
      End
      Begin VB.CommandButton cmdImportConstraints
         Caption         =   "Import Constraints"
         Height          =   495
         Left            =   -70680
         TabIndex        =   6
         ToolTipText     =   "Click here to import a variable/constraint set."
         Top             =   1920
         Width           =   1215
      End
      Begin VB.CommandButton cmdExportConstraints
         Caption         =   "Export Constraints"
         Height          =   495
         Left            =   -70680
      TabIndex      =  5
      ToolTipText   =  "Click here to export a variable/constraint set."
      Top           =  2520
      Width         =  1215
   End
   Begin VB.CommandButton cmdGenerate
      Caption       =  "Generate"
      Height        =  495
      Left          =  -70680
      TabIndex      =  4
      ToolTipText   =  "Click here to generate variants."
      Top           =  720
      Width         =  1215
   End
   Begin VB.TextBox txtVariablize
      BackColor     =  &H8000000C&
      Height        =  375
      Left          =  5880
      TabIndex      =  2
      Text          =  "Rob"
      Top           =  4740
      Visible       =  0   'False
      Width         =  615
   End
   Begin VB.ListBox lstAccepted
      Height        =  2985
      ItemData      =  "TCA.frx":0AD0
      Left          =  240
      List          =  "TCA.frx":0AD2
      MultiSelect   =  2  'Extended
      TabIndex      =  55
      ToolTipText   =  "Left button click on a variant to view it.  Then right button click for
options."
      Top           =  4320
      Width         =  3855
   End
   Begin VB.Label lblAccepted
      Caption       =  "Accepted variants"
      Height        =  255
      Left          =  240
      TabIndex      =  32
      Top           =  4080
      Width         =  2535
   End
   Begin VB.Label lblDiff
```

```
        Caption         =   "Prolog randomization:"
        Height          =   255
        Left            =   -73440
        TabIndex        =   28
        Top             =   840
        Width           =   1935
     End
     Begin VB.Label Label1
        Caption         =   "High"
        Height          =   255
        Left            =   -71760
        TabIndex        =   27
        Top             =   1440
        Width           =   495
     End
     Begin VB.Label lblMed
        Caption         =   "Medium"
        Height          =   255
        Left            =   -72720
        TabIndex        =   26
        Top             =   1440
        Width           =   735
     End
     Begin VB.Label lblLow
        Caption         =   "Low"
        Height          =   255
        Left            =   -73440
        TabIndex        =   25
        Top             =   1440
        Width           =   495
     End
     Begin VB.Label lblDummy
        BorderStyle     =   1  'Fixed Single
        Height          =   375
        Left            =   4680
        TabIndex        =   23
        Top             =   6840
        Visible         =   0   'False
        Width           =   615
     End
     Begin VB.Label lblVariants
        Caption         =   "Variants"
        Height          =   255
        Left            =   -74760
        TabIndex        =   22
```

```
   Top          =  2280
   Width         =  2055
End
Begin ComctlLib.ImageList imlI
   Left          =  4680
   Top           =  7200
   _ExtentX      =  1005
   _ExtentY      =  1005
   BackColor     =  -2147483643
   ImageWidth    =  16
   ImageHeight   =  16
   MaskColor     =  12632256
   _Version      =  327682
   BeginProperty Images {0713E8C2-850A-101B-AFC0-4210102A8DA7}
      NumListImages  =  2
      BeginProperty ListImage1 {0713E8C3-850A-101B-AFC0-4210102A8DA7}
         Picture        =  "TCA.frx":0AD4
         Key            =  ""
      EndProperty
      BeginProperty ListImage2 {0713E8C3-850A-101B-AFC0-4210102A8DA7}
         Picture        =  "TCA.frx":1026
         Key            =  ""
      EndProperty
   EndProperty
End
Begin VB.Label LblNumVariants
   Caption       =  "Number:"
   Height        =  255
   Left          =  -74760
   TabIndex      =  17
   Top           =  900
   Width         =  735
End
Begin VB.Label lblFamily
   Caption       =  "Family members"
   Height        =  255
   Left          =  240
   TabIndex      =  14
   Top           =  540
   Width         =  3615
End
Begin VB.Label lblDistractor
   Caption       =  "Distractor Constraints"
   Height        =  255
   Left          =  -74760
```

```
      TabIndex     =  12
      Top          =  5280
      Width        =  2535
    End
    Begin VB.Label lblCloningConstraints
      Caption      =  "Variation Constraints"
      DragIcon     =  "TCA.frx":1578
      Height       =  255
      Left         =  -74760
      TabIndex     =  11
      Top          =  2880
      Width        =  2535
    End
    Begin VB.Label lblVariables
      Caption      =  "Variables"
      Height       =  255
      Left         =  -74760
      TabIndex     =  9
      Top          =  480
      Width        =  855
    End
  End
  Begin ComctlLib.StatusBar stbS
    Align        =  2  'Align Bottom
    Height       =  300
    Left         =  0
    TabIndex     =  30
    Top          =  8010
    Width        =  11400
    _ExtentX     =  20108
    _ExtentY     =  529
    SimpleText   =  ""
    _Version     =  327682
    BeginProperty Panels {0713E89E-850A-101B-AFC0-4210102A8DA7}
      NumPanels    =  11
      BeginProperty Panel1 {0713E89F-850A-101B-AFC0-4210102A8DA7}
        Alignment    =  2
        AutoSize     =  2
        Bevel        =  0
        Object.Width        =  2117
        MinWidth     =  2117
        Text         =  "Program:"
        TextSave     =  "Program:"
        Key          =  ""
        Object.Tag          =  ""
        EndProperty
        BeginProperty Panel2 {0713E89F-850A-101B-AFC0-4210102A8DA7}
          Alignment     =  1
          AutoSize      =  2
          Object.Width        =  1058
          MinWidth      =  1058
          Key           =  ""
          Object.Tag          =  ""
        EndProperty
        BeginProperty Panel3 {0713E89F-850A-101B-AFC0-4210102A8DA7}
          Alignment     =  2
          AutoSize      =  2
          Bevel         =  0
          Object.Width        =  1773
          MinWidth      =  1764
          Text          =  "Family:"
          TextSave      =  "Family:"
          Key           =  ""
          Object.Tag          =  ""
        EndProperty
        BeginProperty Panel4 {0713E89F-850A-101B-AFC0-4210102A8DA7}
          Alignment     =  1
          AutoSize      =  2
          Object.Width        =  2646
          MinWidth      =  2646
          Key           =  ""
          Object.Tag          =  ""
        EndProperty
        BeginProperty Panel5 {0713E89F-850A-101B-AFC0-4210102A8DA7}
          Alignment     =  2
          AutoSize      =  2
          Bevel         =  0
          Object.Width        =  2117
          MinWidth      =  2117
          Text          =  "Attributes:"
          TextSave      =  "Attributes:"
          Key           =  ""
          Object.Tag          =  ""
        EndProperty
        BeginProperty Panel6 {0713E89F-850A-101B-AFC0-4210102A8DA7}
          Alignment     =  1
          AutoSize      =  2
          Object.Width        =  1058
          MinWidth      =  1058
          Key           =  ""
```

```
    Object.Tag          =  ""
  EndProperty
  BeginProperty Panel7 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment     =  1
    AutoSize      =  2
    Object.Width        =  1058
    MinWidth      =  1058
    Key           =  ""
    Object.Tag          =  ""
  EndProperty
  BeginProperty Panel8 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    AutoSize      =  2
    Object.Width        =  1058
    MinWidth      =  1058
    Key           =  ""
    Object.Tag          =  ""
  EndProperty
  BeginProperty Panel9 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment     =  2
    AutoSize      =  2
    Bevel         =  0
    Object.Width        =  2487
    MinWidth      =  2469
    Text          =  "Active Model:"
    TextSave      =  "Active Model:"
    Key           =  ""
    Object.Tag          =  ""
  EndProperty
  BeginProperty Panel10 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment     =  1
    AutoSize      =  2
    Object.Width        =  450
    MinWidth      =  441
    Key           =  ""
    Object.Tag          =  ""
  EndProperty
  BeginProperty Panel11 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment     =  1
    AutoSize      =  2
    Object.Width        =  2646
    MinWidth      =  2646
    Key           =  ""
    Object.Tag          =  ""
  EndProperty
EndProperty
```

```
End
Begin VB.Menu mnuFile
  Caption        =   "File"
  Begin VB.Menu mnuFileNew
    Caption        =   "New"
  End
  Begin VB.Menu mnuFileOpen
    Caption        =   "Open"
  End
  Begin VB.Menu mnuFileImportItem
    Caption        =   "Import Locked Item"
  End
  Begin VB.Menu mnuFileSaveAs
    Caption        =   "Save As"
    Visible        =   0   'False
  End
  Begin VB.Menu mnuFileSave
    Caption        =   "Save"
    Visible        =   0   'False
  End
  Begin VB.Menu mnuFilePrintSetup
    Caption        =   "Print Setup"
  End
  Begin VB.Menu mnuFileExit
    Caption        =   "Exit"
  End
End
Begin VB.Menu mnuHelp
  Caption        =   "Help"
  NegotiatePosition=   3  'Right
  Begin VB.Menu mnuHelpAbout
    Caption        =   "About"
  End
End
Begin VB.Menu mnuVariables
  Caption        =   "Variables"
  Visible        =   0   'False
  Begin VB.Menu mnuVariablesAdd
    Caption        =   "Add"
  End
  Begin VB.Menu mnuVariablesEdit
    Caption        =   "Edit"
  End
  Begin VB.Menu mnuVariablesRemove
    Caption        =   "Remove"
```

```
      End
      Begin VB.Menu mnuVariablesRemoveAll
        Caption         =   "Remove All"
      End
      Begin VB.Menu mnuVariablesEnableAll
        Caption         =   "Enable All"
      End
      Begin VB.Menu mnuVariablesDisableAll
        Caption         =   "Disable All"
      End
      Begin VB.Menu mnuVariablesTest
        Caption         =   "Test"
      End
    End
    Begin VB.Menu mnuConstraints
      Caption         =   "Constraints"
      Visible         =   0   'False
      Begin VB.Menu mnuConstraintsAdd
        Caption         =   "Add"
      End
      Begin VB.Menu mnuConstraintsEdit
        Caption         =   "Edit"
      End
      Begin VB.Menu mnuConstraintsRemove
        Caption         =   "Remove"
      End
      Begin VB.Menu mnuConstraintsRemoveAll
        Caption         =   "Remove All"
      End
      Begin VB.Menu mnuConstraintsEnableAll
        Caption         =   "Enable All"
      End
      Begin VB.Menu mnuConstraintsDisableAll
        Caption         =   "Disable All"
      End
      Begin VB.Menu mnuConstraintsTest
        Caption         =   "Test"
      End
    End
    Begin VB.Menu mnuDisp
      Caption         =   "Disposition"
      Visible         =   0   'False
      Begin VB.Menu mnuDispAccept
        Caption         =   "Accept"
      End
      Begin VB.Menu mnuDispDefer
        Caption       =  "Defer"
      End
      Begin VB.Menu mnuDispDiscard
        Caption       =  "Discard"
      End
      Begin VB.Menu mnuDispMakeModel
        Caption       =  "Create Model"
      End
   End
   Begin VB.Menu mnuTree
      Caption       =  "Tree"
      Visible       =  0   'False
      Begin VB.Menu mnuTreeExtend
        Caption       =  "Extend"
        Enabled       =  0   'False
      End
      Begin VB.Menu mnuTreeRemove
        Caption       =  "Remove"
      End
   End
   Begin VB.Menu mnuAccepted
      Caption       =  "Accepted"
      Visible       =  0   'False
      Begin VB.Menu mnuAcceptedProfile
        Caption       =  "Edit profile"
      End
      Begin VB.Menu mnuAcceptedCopy
        Caption       =  "Copy profile"
        Enabled       =  0   'False
      End
      Begin VB.Menu mnuAcceptedPaste
        Caption       =  "Paste profile"
        Enabled       =  0   'False
      End
   End
End
Attribute VB_Name = "frmTCA"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

' contains family
```

```
Private mudtFam As Family

' word
Private mudtWord As MSWord

' prolog activex
Private mudtProlog As Prolog

' needed for SetAllCheckboxes sub
Private mlstCurrentListBox As ListBox

' needed so frmConstraint know which kind of constraint to create
Private mintConstrLBInd As Integer

' used as a flag when mnuFileImportLockedItem calls mnuFileNew
Private mudtItemType As ItemType

' holding area for copy / paste of variant profiles
Private mudtClone As Clone

' turn full window drag back on if this is TRUE
Private mblnRestoreFullWindowDrag As Boolean

Public Enum EditFlags
   aeNothing = 0
   aeAdd = 1
   aeEdit = 2
End Enum

Public Enum TestType
   tcTestVariables = 0
   tcTestVariationConstraints = 1
   tcTestDistractorConstraints = 2
   tcTestAll = 4
End Enum

' for importing/exporting variables and constraints
Private Enum ConstraintRecordLayout
   crVariableIndex = 1 ' 4 byte long
   crConstraintIndex = 5 ' 4 byte long
   crVariables = 9 ' binary - variable size
End Enum

Private Enum IconImage
   imSnowflake = 1
   imSun = 2
End Enum

' used to update status bar
Private Enum PanelIndex
   pnProgramCaption = 1
   pnProgramName = 2
   pnFamilyCaption = 3
   pnFamilyName = 4
   pnAttributesCaption = 5
   pnItemType = 6
   pnGeneric = 7
   pnProximity = 8
   pnActiveModelCaption = 9
   pnActiveModelIcon = 10
   pnActiveModelName = 11
End Enum

Public Property Get Family() As Family

   Set Family = mudtFam

End Property

Public Property Let Family(ByVal udtNewValue As Family)

   mudtFam = udtNewValue

End Property

Private Sub cmdCancel_Click()

End Sub

Private Sub cmdAcceptedCopy_Click()

   Call mnuAcceptedCopy_Click

End Sub

Private Sub cmdAcceptedEdit_Click()

   Call mnuAcceptedProfile_Click

End Sub
```

```
Private Sub cmdAcceptedPaste_Click()

    Call mnuAcceptedPaste_Click

End Sub

Private Sub cmdComments_Click()

    frmComments.Comment = mudtFam.ActiveModel.Comments
    frmComments.Show vbModal
    mudtFam.ActiveModel.Comments = frmComments.Comment

    UpdateTab1ControlStates

End Sub

Private Sub cmdConstraintAdd_Click(index As Integer)

    mintConstrLBInd = index
    Call mnuConstraintsAdd_Click

End Sub

Private Sub cmdConstraintEdit_Click(index As Integer)

    mintConstrLBInd = index
    Call mnuConstraintsEdit_Click

End Sub

Private Sub cmdConstraintRemove_Click(index As Integer)

    mintConstrLBInd = index
    Call mnuConstraintsRemove_Click

End Sub

Private Sub cmdConstraintTest_Click(index As Integer)

    mintConstrLBInd = index
    Call mnuConstraintsTest_Click

End Sub

Private Sub cmdDispAccept_Click()
```

```
  Call mnuDispAccept_Click

End Sub

Private Sub cmdDispDefer_Click()

  Call mnuDispDefer_Click

End Sub

Private Sub cmdDispDiscard_Click()

  Call mnuDispDiscard_Click

End Sub

Private Sub cmdDisplayModel_Click()

  Call mudtFam.ActiveModel.OpenDoc(mudtWord)

End Sub

Private Sub cmdDispMakeModel_Click()

  Call mnuDispMakeModel_Click

End Sub

Private Sub cmdDone_Click()

  Dim intI As Integer
  Dim udtClone As Clone
  Dim dMode As String
  Dim iType As String
  Dim key As String
  Dim Program As String
  Dim root As String
  Dim udtFamIni As New IniFile
  Dim udtProgress As New Progress

  If MsgBox("Prepare this family for export to TCS?", _
    vbQuestion + vbYesNo) = vbNo Then
    Exit Sub
  End If
```

```
If mudtFam.ActiveModel Is Nothing Then
   ' do nothing
Else
   mudtFam.ActiveModel.WriteModel
End If

' close this so it can be copied to the out directory
mudtFam.ActiveModel.CloseDoc

Call udtProgress.Init(mudtFam.Clones.Count + 2, "Preparing family for exporting to TCS...")
udtProgress.Advance

root = ExtractFileNameNoExt(mudtFam.FileName)
udtFamIni.FN = OUT_DIRECTORY & root & ".ini"

Select Case mudtFam.Program
   Case prGRE
      Program = "GRE"

   Case prGMAT
      Program = "GMAT"

   Case prSAT
      Program = "SAT"

   Case prMR
      Program = "MR"
End Select

Dim udtInIni As New IniFile

udtInIni.FN = left(mudtFam.FileName, Len(mudtFam.FileName) - 3) & _
   "ini"

Dim strModelNo As String

' started with a locked item (during this session)
strModelNo = udtInIni.GetProfileString("LockedItemData", _
   "TCAModelNo")

' started with an existing family (during this session)
If strModelNo = "Not Found" Then
   strModelNo = udtInIni.GetProfileString("Family", _
      "TCAModelNo")
End If
```

```
Call udtFamIni.SetKeyValuePair("TCAModelNo", strModelNo)
Call udtFamIni.SetKeyValuePair("LockedAccnum", mudtFam.AccNum)
Call udtFamIni.SetKeyValuePair("Program", Program)

Dim strProx As String

Select Case mudtFam.Proximity
   Case prNear
      strProx = "close"
   Case prMedium
      strProx = "medium"
   Case prFar
      strProx = "far"
End Select
Call udtFamIni.SetKeyValuePair("Proximity", strProx)

If mudtFam.Generic Then
   Call udtFamIni.SetKeyValuePair("Nature", "generic")
Else
   Call udtFamIni.SetKeyValuePair("Nature", "non-generic")
End If

For Each udtClone In mudtFam.Clones

   udtClone.CloseDoc

   If udtClone.IsRouted = False Then

      dMode = "TCA"
      iType = "TCA"

      Call FileCopy(IN_DIRECTORY & udtClone.FileName, _
               OUT_DIRECTORY & udtClone.FileName)

   Else

      If udtClone.DeliveryMode = dmPPT Then
        dMode = "PPT"
      Else
        dMode = "CBT"
      End If

      Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
      Select Case mudtFam.ItemType
```

```
        Case ptStandardMC
            If dMode = "PPT" Then
                iType = "MC Item"
                Call genPPT_MultChoice(udtClone, key)
            Else
                iType = "QANTDISC"
                Call genCBT_MultChoice(udtClone, key)
            End If

        Case ptQuantComp
            If dMode = "PPT" Then
                iType = "QC Discrete"
                Call genPPT_QuantComp(udtClone, key)
            Else
                iType = "QANTCOMP"
                Call genCBT_QuantComp(udtClone, key)
            End If

        Case ptDataSuff
            iType = "DATASUFF"
            Call genCBT_DataSuff(udtClone, key)

    End Select

    udtClone.CloneDoc.Close

End If

Dim udtClnIni As New IniFile

root = ExtractFileNameNoExt(udtClone.FileName)
Call udtFamIni.SetKeyValuePair("Variant", root)

udtClnIni.FN = OUT_DIRECTORY & root & ".ini"

Call udtClnIni.SetKeyValuePair("DeliveryMode", dMode)
Call udtClnIni.SetKeyValuePair("Key", udtClone.key)
Call udtClnIni.SetKeyValuePair("ItemType", iType)
Call udtClnIni.WriteProfileSection("Variant")
Call udtClnIni.WriteProfileString("Exit", " ", " ")

Set udtClnIni = Nothing

udtProgress.Advance
```

```
Next udtClone

' delete profiled variants from lstAccepted
With lstAccepted
   intI = .ListCount - 1
   Do While intI > -1
      Set udtClone = mudtFam.Clones.Item(Str(.ItemData(intI)))
      If udtClone.IsRouted Then
         ' remove the clone from the collection
         Call mudtFam.Clones.Remove(Str(.ItemData(intI)))
         ' remove it from the list box
         Call .RemoveItem(intI)
      End If
      intI = intI - 1
   Loop
End With

mudtFam.WriteFamily

Dim fName As String
Dim strWildCard As String

For intI = 1 To treModels.Nodes.Count
   root = ExtractFileNameNoExt(treModels.Nodes.Item(intI))

   fName = root & ".doc"
   Call udtFamIni.SetKeyValuePair("Member", fName)
   Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)

   fName = root & ".mdl"
   Call udtFamIni.SetKeyValuePair("Member", fName)
   Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)

   If intI = 1 Then
      fName = root & ".mdf"
      strWildCard = root & "*.*"
      Call udtFamIni.SetKeyValuePair("Member", fName)
      Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)
   End If

Next

Call udtFamIni.WriteProfileSection("Family")
Call udtFamIni.WriteProfileString("Exit", " ", " ")
```

```
    ClearControls

    mudtWord.WordApp.Documents.Open FileName:=App.path & "\tcaclone.doc"
    mudtWord.WordApp.Documents.Close

    Kill IN_DIRECTORY & strWildCard

    If strModelNo <> "Not Found" Then
      Kill IN_DIRECTORY & strModelNo & ".*"
    End If

    udtProgress.Advance

    UpdateTab0ControlStates

End Sub

Private Sub genPPT_MultChoice(udtClone As Clone, itmKey As String)

    Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

    docTCAModel.Variables.Add "PROP_ACCNUM", "SSMCPPT"

'   mudtWord.WordApp.Run ("SetAccnum")
    mudtWord.WordApp.Run ("StartItem.Main")

    Dim tabchr As String
    tabchr = Chr(9)
    Dim destRange As Range
    Set destRange = docTCAModel.Content
    destRange.find.Style = "PPTStem"
    destRange.find.Execute FindText:=tabchr

'   MsgBox "PPT MultChoice"

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    destRange.Paste
    destRange.Borders.Enable = False
    destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)
    destRange.Style = "PPTStem"

    Dim respRange As Range
    Dim abcde As String
    abcde = "ABCDE"
```

```
Dim i As Integer

For i = 1 To 5

  Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & Mid(abcde, i, 1)).Range
  respRange.start = respRange.start + 4
  respRange.Copy

  Set destRange = docTCAModel.Content
  destRange.find.Style = "PPTOptions"
  destRange.find.Execute FindText:="(" & Mid(abcde, i, 1) & ")"
  destRange.start = destRange.start + 4
  destRange.Paste
  destRange.Borders.Enable = False
  destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)
  destRange.Style = "PPTOptions"

Next

Dim key As String
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
key = Mid(key, 8, 1)
itmKey = key

For i = 1 To 5
  If key = Mid(abcde, i, 1) Then
    key = Format(i)
    Exit For
  End If
Next

Dim keyRange As Range
Dim keyStart As Long
Set keyRange = docTCAModel.Content
keyStart = keyRange.End - 1

docTCAModel.Content.InsertAfter Text:=itmKey
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

Dim tmpFName As String
tmpFName = OUT_DIRECTORY & udtClone.FileName

docTCAModel.Variables("PROP_ACCNUM").Delete
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
```

```
    docTCAModel.SaveAs tmpFName
    docTCAModel.Close

End Sub

Private Sub genCBT_MultChoice(udtClone As Clone, itmKey As String)

    Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

'   MsgBox "CBT MultChoice"

    docTCAModel.Variables.Add "PROP_ACCNUM", "SSMCCBT"
'   mudtWord.WordApp.Run ("SetAccnum")
    mudtWord.WordApp.Run ("StartItem.Main")

    Dim tabchr As String
    tabchr = Chr(9)
    Dim destRange As Range
    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter stem here."

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    destRange.Paste
    destRange.Borders.Enable = False

    Dim respRange As Range
    Dim abcde As String
    abcde = "ABCDE"
    Dim i As Integer

    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter responses here"
    destRange.End = destRange.End + 1
    destRange.Delete

    For i = 1 To 5

      Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & Mid(abcde, i, 1)).Range
      respRange.start = respRange.start + 4
      respRange.Copy

      destRange.Paste
      destRange.Style = "Choice"
      destRange.InsertParagraphAfter
    Set destRange = destRange.Paragraphs(1).Next.Range

  Next

  Dim key As String
  key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
  key = Mid(key, 8, 1)
  itmKey = key

  For i = 1 To 5
    If key = Mid(abcde, i, 1) Then
      key = Format(i)
      Exit For
    End If
  Next

  Dim keyRange As Range
  Dim keyStart As Long
  Set keyRange = docTCAModel.Content
  keyStart = keyRange.End - 1

  docTCAModel.Content.InsertAfter Text:=itmKey
  keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

  Dim tmpFName As String
  tmpFName = OUT_DIRECTORY & udtClone.FileName

  docTCAModel.Variables("PROP_ACCNUM").Delete
  docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
  Call itemKey_Store(docTCAModel, udtClone.key)
  docTCAModel.SaveAs tmpFName
  docTCAModel.Close

End Sub

Private Sub genPPT_QuantComp(udtClone As Clone, itmKey As String)

  Dim docTCAModel As Document
  Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

' MsgBox "PPT QuantComp"

  docTCAModel.Variables.Add "PROP_ACCNUM", "QCPPT"
' mudtWord.WordApp.Run ("SetAccnum")
```

```
mudtWord.WordApp.Run ("StartItem.Main")

udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
docTCAModel.Tables(1).Cell(Row:=1, Column:=2).Range.Paste
docTCAModel.Tables(1).Cell(Row:=1, Column:=2).Range.Style = "PPTQC StimCentered"

udtClone.CloneDoc.Bookmarks("tca_ColumnA").Range.Copy
docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Paste

udtClone.CloneDoc.Bookmarks("tca_ColumnB").Range.Copy
docTCAModel.Tables(1).Cell(Row:=2, Column:=4).Range.Paste

docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Style = "PPTQC AB"
docTCAModel.Tables(1).Cell(Row:=2, Column:=4).Range.Style = "PPTQC AB"

Dim key As String
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
key = Mid(key, 8, 1)
itmKey = key

Dim abcde As String
abcde = "ABCDE"
Dim i As Integer

For i = 1 To 5
  If key = Mid(abcde, i, 1) Then
    key = Format(i)
    Exit For
  End If
Next

Dim keyRange As Range
Dim keyStart As Long
Set keyRange = docTCAModel.Content
keyStart = keyRange.End - 1

docTCAModel.Content.InsertAfter Text:=itmKey
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

Dim tmpFName As String
tmpFName = OUT_DIRECTORY & udtClone.FileName

docTCAModel.Variables("PROP_ACCNUM").Delete
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
```

```
    docTCAModel.SaveAs tmpFName
    docTCAModel.Close

End Sub

Private Sub genCBT_QuantComp(udtClone As Clone, itmKey As String)

    Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

'   MsgBox "CBT QuantComp"

    docTCAModel.Variables.Add "PROP_ACCNUM", "QCCBT"
'   mudtWord.WordApp.Run ("SetAccnum")
    mudtWord.WordApp.Run ("StartItem.Main")

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=1, Column:=1).Range.Paste

    udtClone.CloneDoc.Bookmarks("tca_ColumnA").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=2, Column:=1).Range.Paste

    udtClone.CloneDoc.Bookmarks("tca_ColumnB").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Paste

    Dim key As String
    key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
    key = Mid(key, 8, 1)
    itmKey = key

    Dim abcde As String
    abcde = "ABCDE"
    Dim i As Integer

    For i = 1 To 5
      If key = Mid(abcde, i, 1) Then
        key = Format(i)
        Exit For
      End If
    Next

    Dim keyRange As Range
    Dim keyStart As Long
    Set keyRange = docTCAModel.Content
    keyStart = keyRange.End - 1
    docTCAModel.Content.InsertAfter Text:=itmKey
    keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
'   docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

    Dim tmpFName As String
    tmpFName = OUT_DIRECTORY & udtClone.FileName

    docTCAModel.Variables("PROP_ACCNUM").Delete
    docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
    Call itemKey_Store(docTCAModel, udtClone.key)
    docTCAModel.SaveAs tmpFName
    docTCAModel.Close

End Sub

Private Sub genCBT_DataSuff(udtClone As Clone, itmKey As String)

    Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

'   MsgBox "CBT DataSuff"

    docTCAModel.Variables.Add "PROP_ACCNUM", "DSCBT"
'   mudtWord.WordApp.Run ("SetAccnum")
    mudtWord.WordApp.Run ("StartItem.Main")

    Dim tabchr As String
    tabchr = Chr(9)
    Dim destRange As Range
    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter stem here."

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    destRange.Paste

'   destRange.Borders.Enable = False
'   destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)

    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter Data Sufficiency Statement 1 here, then press
return."

'   udtClone.CloneDoc.Bookmarks("tca_fStatement").Range.Copy
    Dim srcRange As Range
    Set srcRange = udtClone.CloneDoc.Bookmarks("tca_fStatement").Range
```

```
srcRange.End = srcRange.End - 1
If Len(srcRange.Text) > 0 Then
   srcRange.Copy
   destRange.Paste
End If
destRange.Collapse Direction:=wdCollapseEnd
destRange.InsertParagraphAfter
destRange.Collapse Direction:=wdCollapseEnd

Set srcRange = udtClone.CloneDoc.Bookmarks("tca_sStatement").Range
srcRange.End = srcRange.End - 1
If Len(srcRange.Text) > 0 Then
   srcRange.Copy
   destRange.Paste
End If

Dim n As Integer
n = docTCAModel.ListParagraphs.Count
While n > 2
   Set destRange = docTCAModel.ListParagraphs(n).Range
   destRange.Delete
   n = n - 1
Wend

Dim key As String
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
key = Mid(key, 8, 1)
itmKey = key

Dim abcde As String
abcde = "ABCDE"
Dim i As Integer

For i = 1 To 5
  If key = Mid(abcde, i, 1) Then
    key = Format(i)
    Exit For
  End If
Next

Dim keyRange As Range
Dim keyStart As Long
Set keyRange = docTCAModel.Content
keyStart = keyRange.End - 1
```

```
    docTCAModel.Content.InsertAfter Text:=itmKey
    keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
'   docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

    Dim tmpFName As String
    tmpFName = OUT_DIRECTORY & udtClone.FileName

    docTCAModel.Variables("PROP_ACCNUM").Delete
    docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
    Call itemKey_Store(docTCAModel, udtClone.key)
    docTCAModel.SaveAs tmpFName
    docTCAModel.Close

End Sub

Private Sub itemKey_Store(doc As Document, ByVal key As String)

    Dim i As Integer

    For i = 1 To 5
      If key = Mid("ABCDE", i, 1) Then
        key = Format(i)
        Exit For
      End If
    Next

    doc.Variables.Add "ItemKeyStore", key

End Sub

Private Sub cmdPrintConstraints_Click()

    Dim udtV As Variable
    Dim udtC As Constraint
    Dim udtVI As VarInteger
    Dim udtVR As VarReal
    Dim udtVF As VarFraction
    Dim udtVS As VarString
    Dim udtP As New PrintModel
    Dim varS As Variant
    Dim varS1 As Variant
    Dim udtSS As SubString
    Dim intI As Integer

    udtP.ModelName = ExtractFileNameNoExt(mudtFam.ActiveModel.FileName)
```

```
Call udtP.PrintString("Variables:", 1)

For Each udtV In mudtFam.ActiveModel.Variables
   Call udtP.PrintString("Variable name: " & udtV.name, 2)
   Select Case udtV.Typ
      Case vtInteger
         Call udtP.PrintString("Type: Integer", 3)
      Case vtReal
         Call udtP.PrintString("Type: Real", 3)
      Case vtFraction
         Call udtP.PrintString("Type: Fraction", 3)
      Case vtString
         Call udtP.PrintString("Type: String", 3)
      Case vtUntyped
         Call udtP.PrintString("Type: Untyped", 3)
   End Select
   If udtV.Enabled Then
      Call udtP.PrintString("Status: Enabled", 3)
   Else
      Call udtP.PrintString("Status: Disabled", 3)
   End If
   If udtV.Checksum Then
      Call udtP.PrintString("Checksum: Enabled", 3)
   Else
      Call udtP.PrintString("Checksum: Disabled", 3)
   End If
   Select Case udtV.Typ
      Case vtInteger
         Set udtVI = udtV
         If udtVI.IsIndependent Then
            Call udtP.PrintString("Is independent = True," & _
               " Range: from " & udtVI.From & _
               " to " & udtVI.Too & _
               " by " & udtVI.By, 3)
         Else
            Call udtP.PrintString("Is independent = False", 3)
         End If
      Case vtReal
         Set udtVR = udtV
         If udtVR.IsIndependent Then
            Call udtP.PrintString("Is independent = True," & _
               " Range: from " & udtVR.From & _
               " to " & udtVR.Too & _
               " by " & udtVR.By, 3)
```

```
        Else
          Call udtP.PrintString("Is independent = False", 3)
        End If
        If udtVR.IsOnGrid Then
          Call udtP.PrintString("Force on grid value: True", 3)
        Else
          Call udtP.PrintString("Force on grid value: False", 3)
        End If
        Call udtP.PrintString("# Decimal places: " & _
          Str(udtVR.DecimalPlaces), 3)
        If udtVR.TrailingZeros Then
          Call udtP.PrintString("Display trailing zeros: True", 3)
        Else
          Call udtP.PrintString("Display trailing zeros: False", 3)
        End If
      Case vtFraction
        Set udtVF = udtV
        If udtVF.IsIndependent Then
          Call udtP.PrintString("Is independent = True," & _
            " Range: from " & udtVF.FromNumerator & _
            "/" & udtVF.FromDenominator & _
            " to " & udtVF.ToNumerator & _
            "/" & udtVF.ToDenominator & _
            " by " & udtVF.ByNumerator & _
            "/" & udtVF.ByDenominator, 3)
        Else
          Call udtP.PrintString("Is independent = False", 3)
        End If
        If udtVF.MixedNumbers Then
          Call udtP.PrintString("Display mixed number: True", 3)
        Else
          Call udtP.PrintString("Display mixed number: False", 3)
        End If
      Case vtString
        Set udtVS = udtV
        If udtVS.IsIndexed Then
          Call udtP.PrintString("Indexed: True", 3)
          Call udtP.PrintString("Value Sets:", 3)
          For Each varS In udtVS.StringCollection
            Set udtSS = New SubString
            udtSS.Delimiter = Chr(STRING_DELIMITER)
            udtSS.StringValue = varS
            Call udtP.PrintString("Values:", 4)
            intI = 1
            For Each varS1 In udtSS.StringCollection
```

```
                Call udtP.PrintString(Str(intI) & ". " & varS1, 5)
                intI = intI + 1
              Next varS1
            Next varS
          Else
            Call udtP.PrintString("Indexed: False", 3)
            Call udtP.PrintString("Values:", 3)
            For Each varS In udtVS.StringCollection
              Call udtP.PrintString(varS, 4)
            Next varS
          End If
        Case vtUntyped
      End Select

  Next udtV

  Call udtP.PrintString("Constraints:", 1)

  Call udtP.PrintString("Variation constraints:", 2)

  For Each udtC In mudtFam.ActiveModel.Constraints
    If udtC.ConstraintType = ctVariation Then
      Call udtP.PrintString("Constraint: " & udtC.ConstraintString, 3)
      If udtC.Enabled Then
        Call udtP.PrintString("Status: Enabled", 4)
      Else
        Call udtP.PrintString("Status: Disabled", 4)
      End If
    End If
  Next udtC

  ' exit if not MC
  If Not mudtFam.ItemType = ptStandardMC Then Exit Sub

  Call udtP.PrintString("Distractor constraints:", 2)

  For Each udtC In mudtFam.ActiveModel.Constraints
    If udtC.ConstraintType = ctDistractor Then
      Call udtP.PrintString("Constraint: " & udtC.ConstraintString, 3)
      If udtC.Enabled Then
        Call udtP.PrintString("Status: Enabled", 4)
      Else
        Call udtP.PrintString("Status: Disabled", 4)
      End If
    End If
    Next udtC

End Sub

Private Sub cmdSetAttributes_Click()

    frmAttributes.Show vbModal

    If frmAttributes.OK Then
        mudtFam.Generic = frmAttributes.Generic
        mudtFam.Proximity = frmAttributes.Proximity
        mudtFam.IsDirty = True
        ' save family
        mudtFam.WriteFamily
        UpdateFamilyAttributes
    End If

End Sub

Private Sub cmdTreeExtend_Click()

    Call mnuTreeExtend_Click

End Sub

Private Sub cmdTreeRemove_Click()

    Call mnuTreeRemove_Click

End Sub

Private Sub cmdVariableAdd_Click()

    Call mnuVariablesAdd_Click
    frmVariable.Model = mudtFam.ActiveModel
    frmVariable.ListBox = lstVariables

    frmVariable.Show vbModal

    UpdateTab1ControlStates

End Sub

Private Sub cmdVariableEdit_Click()
```

```
    Call mnuVariablesEdit_Click
    frmVariable.Model = mudtFam.ActiveModel
    frmVariable.ListBox = lstVariables

    If lstVariables.ListIndex >= 0 Then ' Make sure list item is selected
       ' Set the key for access by frmVariable
       With lstVariables
          frmVariable.Variable = _
             mudtFam.ActiveModel.Variables.Item(Str(.ItemData(.ListIndex)))
       End With
       frmVariable.Show vbModal
    End If

    UpdateTab1ControlStates

End Sub

Private Sub cmdVariableRemove_Click()

    Call mnuVariablesRemove_Click

End Sub

Private Sub cmdVariableTest_Click()

    Call mnuVariablesTest_Click

End Sub

Private Sub Form_Initialize()

    frmSplash.Show

End Sub

Private Sub Form_Load()

    ' to trap cancels
    cdlCD.CancelError = True

    ' Create Word Object
    Set mudtWord = New MSWord

    ' get rid of the kill file if it exists, as it will prevent
    ' StartProlog from working
    DestroyKillFile

    ' Create the Prolog object
    If mudtProlog Is Nothing Then
       Set mudtProlog = CreateObject("AXProlog.Prolog")
       If Not mudtProlog.StartProlog Then
          Call MsgBox("Prolog cannot be started.", vbExclamation, "Prolog error")
       End If
    End If

    treModels.ImageList = imlI

    frmSplash.UnloadMe

    Me.Show

    UpdateTab0ControlStates

'   ' copies ied files from a holding area, as TCS deletes them for
'   ' reasons unknown.
'   Call Kill("c:\tcs\working\*.ied")
'   Call FileCopy("c:\tcs\tcaied\dscbt.ied", "c:\tcs\working\dscbt.ied")
'   Call Shell("attrib -r c:\tcs\working\dscbt.ied", vbHide)
'   Call FileCopy("c:\tcs\tcaied\qccbt.ied", "c:\tcs\working\qccbt.ied")
'   Call FileCopy("c:\tcs\tcaied\qcppt.ied", "c:\tcs\working\qcppt.ied")
'   Call FileCopy("c:\tcs\tcaied\ssmccbt.ied", "c:\tcs\working\ssmccbt.ied")
'   Call FileCopy("c:\tcs\tcaied\ssmcppt.ied", "c:\tcs\working\ssmcppt.ied")

End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)

    Call sstMainTab_MouseMove(Button, Shift, X, Y)

End Sub

Private Sub Form_Resize()

    ' if minimized, don't resize
    If Me.WindowState = vbMinimized Then Exit Sub

    Dim udtW As New Win32API
    Dim result As Long

    'Turn off full window drag if it's on
    If udtW.IsFullWindowDragOn Then
        udtW.TurnOffFullWindowDrag
        mblnRestoreFullWindowDrag = True
    End If

    ' adjust horizontals
    fraWord.left = 120
    fraWord.Width = Me.Width - sstMainTab.Width - 360
    sstMainTab.left = fraWord.Width + 180

    'adjust verticals
    fraWord.Height = Me.Height - fraWord.top - stbS.Height - 700 ' approx title bar height
    sstMainTab.Height = fraWord.Height

    mudtWord.Resize

End Sub

Private Sub Form_Unload(Cancel As Integer)

    ' if no active family, hit the road
    If mudtFam Is Nothing Then
        ' do nothing
    Else
        mudtFam.WriteFamily
        If mudtFam.ActiveModel Is Nothing Then  ' see if an active model has been set
            ' do nothing
        Else
            mudtFam.ActiveModel.CloseDoc
            KillVariants  ' Get rid of any variants left on tab 3
            mudtFam.ActiveModel.WriteModel ' save the active model
        End If
    End If

    ' close all docs
    mudtWord.CloseAllDocs

    ' Kill Word before frmTCA is unloaded to prevent automation error
    Set mudtWord = Nothing

    ' force event
    Call sstMainTab_MouseMove(1, 1, 1, 1)

'   To cleanly shut down AXProlog on W95, 98 boxes
    mudtProlog.EndProlog
    ' End required by NT 4.0 to shut down TCA successfully!
    End

End Sub

Private Sub lstVariables_ItemCheck(Item As Integer)

    With lstVariables
        If lstVariables.ListCount = 0 Then Exit Sub ' this prevents an error
        If mudtFam.ActiveModel.IsFrozen Then
            .Selected(Item) = _
              mudtFam.ActiveModel.Variables.Item(Str(.ItemData(Item))).Enabled
        Else
          mudtFam.ActiveModel.Variables.Item(Str(.ItemData(Item))).Enabled = _
              .Selected(Item)
        End If
    End With

    UpdateTab1ControlStates

End Sub

Private Sub lstVariables_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

    Dim strIndex As String

    Set mlstCurrentListBox = lstVariables

    If Button = vbRightButton Then
        frmVariable.AddEditFlag = aeNothing
        PopupMenu mnuVariables ' Pull up popup menu for variable window
        frmVariable.Model = mudtFam.ActiveModel
        frmVariable.ListBox = lstVariables
        Select Case frmVariable.AddEditFlag
          Case aeEdit
            If lstVariables.ListIndex >= 0 Then ' Make sure list item is selected
              ' Set the key for access by frmVariable
              With lstVariables
                frmVariable.Variable = _
                  mudtFam.ActiveModel.Variables.Item(Str(.ItemData(.ListIndex)))
              End With
              frmVariable.Show vbModal
            End If
          Case aeAdd
```

```
                frmVariable.Show vbModal
        End Select
    End If

End Sub

Private Sub lstConstraints_ItemCheck(index As Integer, Item As Integer)

    Dim strKey As String

    With lstConstraints(index)
        If .ListCount = 0 Then Exit Sub ' prevents error if listbox is empty
        If mudtFam.ActiveModel.IsFrozen Then
            .Selected(Item) = _
              mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(Item))).Enabled
        Else
            mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(Item))).Enabled = _
              .Selected(Item)
        End If
    End With

    UpdateTab1ControlStates

End Sub

' provide right button menu options
Private Sub lstConstraints_MouseDown(index As Integer, Button As Integer, _
    Shift As Integer, X As Single, Y As Single)

    Dim strIndex As String

    Set mlstCurrentListBox = lstConstraints(index)
    mintConstrLBInd = index

    Call UpdateTab1ControlStates(index)

    If Button = vbRightButton Then
        PopupMenu mnuConstraints
    Else
        If mudtFam.ActiveModel.IsFrozen = False Then
            lstConstraints(index).Drag
        End If
    End If

End Sub
```

```
' Enable drag and drop between constraint list boxes
Private Sub lstConstraints_DragDrop(index As Integer, Source As Control, _
   X As Single, Y As Single)

   If Source.ListCount = 0 Then
      Exit Sub
   End If

   If index <> Source.index Then ' Assure that it's another listbox!

      Dim udtConstraint As Constraint
      Dim strKey As String

      strKey = Str(Source.ItemData(Source.ListIndex))

      With lstConstraints(index)
         ' Add the dragged constraint to the end of the target listbox
         .List(.ListCount) = Source.List(Source.ListIndex)
         ' Update the index in the new listbox entry
         .ItemData(.ListCount - 1) = Source.ItemData(Source.ListIndex)
      End With

      ' Find the constraint object being moved and update it's "type" in the collection
      Set udtConstraint = mudtFam.ActiveModel.Constraints.Item(strKey)
      udtConstraint.ConstraintType = index

      ' Delete the dragged constraint from the source listbox
      Call Source.RemoveItem(Source.ListIndex)

   End If

   UpdateTab1ControlStates

End Sub

Private Sub lstDisposition_MouseDown(Button As Integer, Shift As Integer, _
   X As Single, Y As Single)

   Dim udtClone As Clone

   If Button = vbRightButton Then
      PopupMenu mnuDisp
   Else
      With lstDisposition
         If .ListCount > 0 Then ' a valid selection has been made
```

```
                Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(.ListIndex)))
                Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
            End If
        End With
    End If
End Sub

Private Sub lstAccepted_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)

    Static udtClone As Clone

    If Button = vbRightButton Then
'       With lstAccepted
'           If .SelCount = 1 Then
'               mnuAcceptedProfile.Enabled = True
'               mnuAcceptedCopy.Enabled = True
'               Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
''               Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
'               Set udtClone = Nothing
'           Else
'               mnuAcceptedProfile.Enabled = False
'               mnuAcceptedCopy.Enabled = False
'           End If
'       End With
        PopupMenu mnuAccepted
    Else ' left button click
        If udtClone Is Nothing Then
            ' do nothing
        Else
            udtClone.CloseDoc
            Set udtClone = Nothing
        End If
        With lstAccepted
            If .ListCount > 0 Then
                Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
                Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
            End If
        End With
    End If

    UpdateTab0ControlStates

End Sub
```

```
Private Sub cmdSaveModel_Click()

  If mudtFam.ActiveModel.IsDirty Then
    mudtFam.ActiveModel.WriteModel
    KillVariants 'delete any variants on tab 3
  End If

  UpdateTab1ControlStates

End Sub

Private Sub cmdTestAll_Click()

  cmdSaveModel_Click ' force a save
  Call TestConstraints(tcTestAll)

End Sub

Private Sub cmdImportConstraints_Click()

  Dim strFN As String

  With cdlCD
    .FileName = ""
    .CancelError = True
    .DialogTitle = "Import constraints from file"
    .Filter = "Constraint Files (*.con)|*.con|"
    .DefaultExt = ".con"
    .InitDir = "c:\tcs\tca\constraints"
    .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
    On Error GoTo Cancel ' trap the Cancel button
    .ShowOpen
    On Error GoTo 0 ' reset the error
    strFN = .FileName
  End With

  ' exit if there's no file name

  If Len(strFN) = 0 Then
    Exit Sub
  End If

  ' create a new collection of imported variables

  Dim udtCVariables As New CVariables
```

```
Call udtCVariables.ReadCollection(strFN, crVariableIndex, crConstraintIndex)

' add the imported variables to the main collection

Dim udtNewVar As Variable

For Each udtNewVar In udtCVariables
   If mudtFam.ActiveModel.Variables.UniqueName(udtNewVar.name) Then
      Call mudtFam.ActiveModel.Variables.AddObject(udtNewVar)
      With lstVariables
         ' Add the new variable to the variable list box
         Call .AddItem(udtNewVar.ScreenFormat)
         ' Set ItemData to index value of the variable object
         .ItemData(.ListCount - 1) = udtNewVar.index
         ' Set the check box.
         .Selected(.ListCount - 1) = udtNewVar.Enabled
      End With
   Else
      Call MsgBox("Variable " & udtNewVar.name & " will not be imported.", _
         vbExclamation, "Variable not unique")
   End If

Next udtNewVar

' read the imported constraints into a new collection

Dim udtCConstraints As New CConstraints

Call udtCConstraints.ReadCollection(strFN, crConstraintIndex, READ_UNTIL_EOF)

' add the imported constraints

Dim udtNewCon As Constraint

For Each udtNewCon In udtCConstraints
   If mudtFam.ActiveModel.Constraints.UniqueConstraint(udtNewCon.ConstraintString)
Then
      Call mudtFam.ActiveModel.Constraints.AddObject(udtNewCon)
      With lstConstraints(udtNewCon.ConstraintType)
         ' Add the new variable to the variable list box
         Call .AddItem(udtNewCon.ConstraintString)
         ' Set ItemData to index value of the variable object
         .ItemData(.ListCount - 1) = udtNewCon.index
         ' Check the check box
         .Selected(.ListCount - 1) = udtNewCon.Enabled
```

```
        End With
      Else
        Call MsgBox("Constraint " & udtNewCon.ConstraintString & " will not be imported.", _
          vbExclamation, "Constraint not unique")
      End If
    Next udtNewCon

Cancel:
    Exit Sub

End Sub

Private Sub cmdExportConstraints_Click()

    Dim strFN As String

    With cdlCD
      .FileName = ""
      .DialogTitle = "Export constraints to file"
      .Filter = "Constraint Files (*.con)|*.con|"
      .DefaultExt = ".con"
      .InitDir = "c:\tcs\tca\constraints"
      .Flags = cdlOFNOverwritePrompt + cdlOFNHideReadOnly
      On Error GoTo Cancel ' trap the Cancel button
      .ShowSave
      On Error GoTo 0 ' reset
      strFN = .FileName
    End With

    Dim lngEndPos As Long

    If Len(strFN) > 0 Then
      lngEndPos = mudtFam.ActiveModel.Variables.WriteCollection(strFN, crVariableIndex,
crVariables)
      Call mudtFam.ActiveModel.Constraints.WriteCollection(strFN, crConstraintIndex,
lngEndPos)
    End If

Cancel:
    Exit Sub

End Sub

Private Sub cmdPrintBatch_Click()
```

```
    Dim blnTF As Boolean
    Dim udtClone As Clone

    If mudtWord.WordApp.Documents.Count = 0 Then
      mudtWord.WordApp.Documents.Open FileName:=App.path & "\printing.doc"
      blnTF = True
    End If

    For Each udtClone In mudtFam.Clones
      mudtWord.WordApp.PrintOut FileName:=IN_DIRECTORY & udtClone.FileName
    Next udtClone

    If blnTF Then
      mudtWord.WordApp.Documents.Close
    End If

  End Sub

  Private Sub cmdPrintVariants_Click()

    Dim blnTF As Boolean
    Dim udtClone As Clone

    If mudtWord.WordApp.Documents.Count = 0 Then
      mudtWord.WordApp.Documents.Open FileName:=App.path & "\printing.doc"
      blnTF = True
    End If

    For Each udtClone In mudtFam.ActiveModel.Clones
      mudtWord.WordApp.PrintOut FileName:=IN_DIRECTORY & udtClone.FileName
    Next

    If blnTF Then
      mudtWord.WordApp.Documents.Close
    End If

  End Sub

  Private Sub cmdGenerate_Click()

    Dim udtClone As New Clone

    Me.Enabled = False ' disable frmTCA to make next form seem modal
    frmProlog.Caption = "Generating " & txtNum2Generate & " variants"
    frmProlog.lblProlog.Caption = "Click Abort to terminate variant generation."
    frmProlog.Show ' show form modeless so execution continues
    Me.MousePointer = vbHourglass
    Call mudtFam.ActiveModel.GenerateClones(mudtWord, mudtProlog, _
      CInt(txtNum2Generate), sldDifference)
    Me.MousePointer = vbDefault
    frmProlog.Kill ' destroy frmProlog
    Me.Enabled = True

    If lstDisposition.ListCount > 0 Then
      With lstDisposition
        .Selected(.ListCount - 1) = True
        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(.ListCount - 1)))
        Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
      End With
    End If

    UpdateTab2ControlStates

End Sub

Private Sub mnuDispAccept_Click()

    Dim udtClone As Clone
    Dim nodN As Node
    Dim intI As Integer
    Dim strFN As String

    With lstDisposition
      If .SelCount > 0 Then ' make sure something's selected
        For intI = 0 To .ListCount - 1 ' for multiselect
          If .Selected(intI) Then
            strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa
me)
            ' confirm this operation
            If MsgBox("Accept variant " & strFN & "?", _
                vbQuestion + vbYesNo, "Confirm") = vbNo Then
              .Selected(intI) = False
            End If
          End If
          If .Selected(intI) Then
            ' get object from active model's clone collection
            Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
            ' close the document, if it's open
            udtClone.CloseDoc
```

```
                ' remove it from the active model's collection
                Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
                ' save the checksum in the model
                Call mudtFam.ActiveModel.AddChecksum(udtClone.Checksum)
                ' add it to the family clone collection
                Call mudtFam.Clones.AddObj(udtClone)
                ' add it to the accepted list box
                Call lstAccepted.AddItem(ExtractFileName(udtClone.FileName))
                ' add key to itemdata
                lstAccepted.ItemData(lstAccepted.ListCount - 1) = udtClone.index
                ' freeze the model
                mudtFam.ActiveModel.FreezeModel
                ' update the icon
                Set nodN = treModels.Nodes.Item(ModelKey(mudtFam.ActiveModel.FileName))
                nodN.Image = imSnowflake
                stbS.Panels(pnActiveModelIcon).Picture = imlI.ListImages(nodN.Image).Picture
                Call mudtFam.ActiveModel.CloseDoc
                Call mudtFam.ActiveModel.OpenDoc(mudtWord)
             End If
          Next intI
          For intI = .ListCount - 1 To 0 Step -1
             If .Selected(intI) Then
                ' remove the entry from the disposition list box
                Call .RemoveItem(intI)
             End If
          Next intI
       End If
    End With

    UpdateTab0ControlStates
    UpdateTab1ControlStates
    UpdateTab2ControlStates

End Sub

Private Sub mnuDispDefer_Click()

    Dim udtClone As Clone
    Dim intI As Integer
    Dim strFN As String

    With lstDisposition
       If .SelCount > 0 Then ' make sure somethings selected
          For intI = 0 To .ListCount - 1 ' for multiselect
             If .Selected(intI) Then
```

```
                    strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa
me)
                    ' confirm this operation
                    If MsgBox("Defer variant " & strFN & "?", _
                      vbQuestion + vbYesNo, "Confirm") = vbNo Then
                      .Selected(intI) = False
                    End If
                  End If
                  If .Selected(intI) Then
                    ' get object from active model's clone collection
                    Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
                    ' close the document
                    udtClone.CloseDoc
                    ' delete the clone file
                    Kill IN_DIRECTORY & udtClone.FileName
                    ' remove the clone from the active model's collection
                    Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
                  End If
                Next intI
                For intI = .ListCount - 1 To 0 Step -1 ' for multiselect
                  If .Selected(intI) Then
                    ' remove the entry from the disposition list box
                    Call .RemoveItem(intI)
                  End If
                Next intI
            End If
        End With

        UpdateTab2ControlStates

    End Sub

    Private Sub mnuDispDiscard_Click()

        Dim udtClone As Clone
        Dim intI As Integer
        Dim strFN As String

        With lstDisposition
            If .SelCount > 0 Then ' make sure somethings selected
                For intI = 0 To .ListCount - 1 ' for multiselect
                  If .Selected(intI) Then
                    strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa
me)
                    ' confirm this operation
                    If MsgBox("Discard variant " & strFN & "?", _
                       vbQuestion + vbYesNo, "Confirm") = vbNo Then
                       .Selected(intI) = False
                    End If
                 End If
                 If .Selected(intI) Then
                    ' get object from active model's clone collection
                    Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
                    ' save the checksum in the model
                    Call mudtFam.ActiveModel.AddChecksum(udtClone.Checksum)
                    ' close the document
                    udtClone.CloseDoc
                    ' delete the clone file
                    Kill IN_DIRECTORY & udtClone.FileName
                    ' remove the clone from the active model's collection
                    Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
                 End If
              Next intI
              For intI = .ListCount - 1 To 0 Step -1 ' for multiselect
                 If .Selected(intI) Then
                    ' remove the entry from the disposition list box
                    Call .RemoveItem(intI)
                 End If
              Next intI
           End If
        End With

        UpdateTab2ControlStates

     End Sub

     Private Sub mnuDispMakeModel_Click()

        Dim udtClone As Clone
        Dim strNewFN As String
        Dim strKey As String
        Dim strNewKey As String
        Dim udtM As Model
        Dim nodN As Node
        Dim intI As Integer
        Dim strFN As String

        With lstDisposition
        If .SelCount > 0 Then ' make sure somethings selected
          For intI = 0 To .ListCount - 1 ' for multiselect
            If .Selected(intI) Then
              strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa
me)
              ' confirm this operation
              If MsgBox("Create a new model from variant " & strFN & "?", _
                vbQuestion + vbYesNo, "Confirm") = vbNo Then
                .Selected(intI) = False
              End If
            End If
            If .Selected(intI) Then
              ' get object from active model's clone collection
              Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
              ' close the document
              udtClone.CloseDoc
              strKey = ModelKey(udtClone.FileName)
              ' find the next key for this parent model
              strNewKey = NextModelKey(udtClone.FileName)
              ' add the child to the tree
              strNewFN = ModelEmbedKey(udtClone.FileName, strNewKey)
              Set nodN = treModels.Nodes.Add(strKey, tvwChild, strNewKey, strNewFN)
              nodN.Expanded = True
              nodN.sorted = True
              nodN.Image = imSun
              ' copy the clone to the new model file name
              Call FileCopy(IN_DIRECTORY & udtClone.FileName, IN_DIRECTORY &
strNewFN)
              ' make a copy of the parent's model file for this child
              Call FileCopy(ModelFileName(IN_DIRECTORY &
ModelEmbedKey(udtClone.FileName, strKey)), _
                ModelFileName(IN_DIRECTORY & strNewFN))
              ' add the child's model to the model collection. "Thaw" the child.
              Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & strNewFN, _
                mudtFam.ItemType)
              udtM.IsFrozen = False
              ' reset the clone index of the child
              udtM.LastClone = 0
              ' save it
              udtM.WriteModel
              ' tell 'em about it
              Call MsgBox("Variant " & udtClone.FileName & " has been copied to " &
strNewFN, _
                vbInformation, "Model Created")
            End If
          Next intI
        End If
      End With

      UpdateTab0ControlStates
      UpdateTab2ControlStates

    End Sub

    Private Sub mnuFileNew_Click()

      Dim udtWAPI As New Win32API
      Dim strFN As String
      Dim udtProgram As Program
      Dim udtItemType As ItemType
      Dim udtProximity As Proximity
      Dim blnGeneric As Boolean
      Dim udtIni As New IniFile

      ' clear out everything
      ClearControls

      ' get family values (pun intended)
      frmNew.Show vbModal
      If frmNew.OK = False Then GoTo Cancel

      udtProgram = frmNew.Program
      udtItemType = frmNew.ItemType
      udtProximity = frmNew.Proximity
      blnGeneric = frmNew.Generic

      With cdlCD
        .InitDir = IN_DIRECTORY
        .FileName = ""
        .DialogTitle = "Save new family as"
        .Filter = "Model Doc Files (*$R.doc)|*$R.doc|"
        .DefaultExt = ".doc"
        .Flags = cdlOFNHideReadOnly
        On Error GoTo Cancel
        .ShowSave
        On Error GoTo 0
        strFN = .FileName
      End With
```

```
' see if an FN was entered
If Len(strFN) = 0 Then
   Beep
   GoTo Cancel
End If

strFN = UCase(strFN)

' don't allow family to be created if it's not in the "IN" directory
If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
   ' do nothing
Else
   Call MsgBox("Family must be located in " & IN_DIRECTORY, _
      vbExclamation, "Error")
   GoTo Cancel
End If

' check the extension
If (InStr(1, strFN, ".doc", vbTextCompare)) = 0 Then
   Call MsgBox("Invalid file name extension.", vbExclamation, "Error")
   GoTo Cancel
End If

Dim varI As Variant

' embed $R into FN if the user hasn't
If InStr(1, strFN, "$R.doc", vbTextCompare) = 0 Then
   varI = InStr(1, strFN, ".doc", vbTextCompare)
   strFN = Mid(strFN, 1, varI - 1) & "$R.doc"
End If

' check for unique FN
If udtWAPI.FileExists(strFN) Then
   Call MsgBox("File name " & _
      ExtractFileName(strFN) & " is not unique.", _
      vbExclamation, "Error")
   GoTo Cancel
End If

Dim strShortFN As String
strShortFN = ExtractFileName(strFN)

' create a new family object
Set mudtFam = New Family
```

```
' set file name, program, and item type
mudtFam.FileName = strFN
mudtFam.Program = udtProgram
mudtFam.ItemType = udtItemType
mudtFam.Proximity = udtProximity
mudtFam.Generic = blnGeneric
mudtFam.IsDirty = True

' put the family name on the status bar
stbS.Panels(pnFamilyName) = strShortFN

' fill in the rest of the status bar
UpdateFamilyAttributes

' format tab 2
Call FormatTab2(mudtFam.ItemType)

' copy correct Word template to new model FN
Select Case mudtFam.ItemType

  Case ptStandardMC
    FileCopy App.path & "\TCASMC.doc", strFN

  Case ptQuantComp
    FileCopy App.path & "\TCAQC.doc", strFN

  Case ptDataSuff
    FileCopy App.path & "\TCADS.doc", strFN

End Select

Dim nodN As Node

' clear out the treeview box
treModels.Nodes.Clear

' add the new root
Set nodN = treModels.Nodes.Add(, , "R", strShortFN, imSun)
nodN.Expanded = True
nodN.sorted = True
nodN.Selected = True

Call mudtFam.Models.AddNew(strFN, mudtFam.ItemType)

' enable attributes button
```

```
    cmdSetAttributes.Enabled = True

    ' force event to set active model
    treModels_Click

Cancel:
    UpdateTab0ControlStates

    Exit Sub

End Sub

Private Sub mnuFileOpen_Click()

    Dim strFN As String

    ' clear out everything
    ClearControls

    With cdlCD
        .InitDir = IN_DIRECTORY
        .FileName = ""
        .CancelError = True
        .DialogTitle = "Open model root"
        .Filter = "Model Doc Files (*$R.doc)|*$R.doc|"
        .DefaultExt = ".doc"
        .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
        On Error GoTo Cancel
        .ShowOpen
        On Error GoTo 0
        strFN = .FileName
    End With

    ' exit if there's no file name
    If Len(strFN) = 0 Then
        Exit Sub
    End If

    strFN = UCase(strFN)

    ' don't allow family to be opened if it's not in the "IN" directory
    If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
        ' do nothing
    Else
```

```
        Call MsgBox("Family must be located in " & IN_DIRECTORY, _
            vbExclamation, "Error")
        Exit Sub
    End If

    ' find all of the children
    Dim nodN As Node
    Dim strIndex As String
    Dim strT As String
    Dim varI1 As Variant
    Dim udtWAPI As New Win32API
    Dim strNewFN As String
    Dim colFN As Collection

    ' add a wild card to the file name
    varI1 = InStr(1, strFN, ".")
    strNewFN = Mid(strFN, 1, varI1 - 1) & "*" & Mid(strFN, varI1, _
        Len(strFN) - varI1 + 1)

    ' get a collection of file names (*.doc) matching the wild card
    Set colFN = udtWAPI.FindAllFiles(strNewFN)

    ' create a new family object
    Set mudtFam = New Family

    Dim strMdfFN As String

    ' make sure the .mdf file is there.
    strMdfFN = left(strFN, Len(strFN) - 3) & "mdf"
    If udtWAPI.FileExists(strMdfFN) = False Then
        Call MsgBox("This family has a " & _
            "missing mdf file and cannot be loaded.  " & _
            "File " & strMdfFN & " is not in the IN directory.", _
            vbExclamation, "Error")
        Exit Sub
    End If

    ' set the file name of the family, read.
    mudtFam.FileName = strFN
    mudtFam.ReadFamily

    Dim udtClone As Clone

    ' verify that all variants referenced in the family object are in
    ' the IN directory.
```

```
For Each udtClone In mudtFam.Clones
   ' the next line allows families to be renamed between TCA sessions
   udtClone.FileName = ExtractFamilyName(strFN) & _
      ExtractFamilyKey(udtClone.FileName) & ".doc"
   If udtWAPI.FileExists(IN_DIRECTORY & udtClone.FileName) = False Then
      Call MsgBox("This family has at least " & _
         "one missing variant file and cannot be loaded.  " & _
         "File " & udtClone.FileName & " is not in the IN directory.", _
         vbExclamation, "Error")
      Exit Sub
   End If
Next udtClone

' put family name on status bar
stbS.Panels(pnFamilyName) = ExtractFileName(strFN)

' format tab 2
Call FormatTab2(mudtFam.ItemType)

' update the accepted listbox with leftover clones
For Each udtClone In mudtFam.Clones
   With lstAccepted
      If udtClone.IsRouted Then
         Call .AddItem(udtClone.FileName & ": Routed to TCS")
      Else
         Call .AddItem(udtClone.FileName)
      End If
      .ItemData(.ListCount - 1) = udtClone.index
   End With
Next udtClone

' select the first entry, if there is one
If lstAccepted.ListCount > 0 Then
   lstAccepted.Selected(0) = True
End If

' display attribute info on status bar
UpdateFamilyAttributes

' clear out the dummy list box
Call lstDummy.Clear

Dim varFN As Variant
Dim udtM As Model
Dim intI As Integer
Dim intIcon As Integer

' dump the file names into a dummy list box which will sort them automatically.
' the tree control must add them in heirarchical order.

For Each varFN In colFN
   varI1 = InStr(1, varFN, ".")
   If IsNumeric(Mid(varFN, varI1 - 1, 1)) = False Then ' it's not a clone
      Call lstDummy.AddItem(varFN) ' add the model
   End If
Next varFN

Dim strMdlFN As String

For intI = 0 To lstDummy.ListCount - 1

   varFN = lstDummy.List(intI)
   strIndex = ModelKey(varFN)
   If UCase(strIndex) = "R" Then
      Set nodN = treModels.Nodes.Add(, , strIndex, varFN)
      Set treModels.SelectedItem = nodN
   Else
      Set nodN = treModels.Nodes.Add(left(strIndex, Len(strIndex) - 1), _
         tvwChild, strIndex, varFN)
   End If
   ' test to see if corresponding .mdl file exists
   strMdlFN = left(varFN, Len(varFN) - 3) & "mdl"
   If udtWAPI.FileExists(strMdlFN) = False Then
      Call MsgBox("This family has at least " & _
         "one missing mdl file and cannot be loaded.  " & _
         "File " & strMdlFN & " is not in the IN directory.", _
         vbExclamation, "Error")
      ClearControls
      Exit Sub
   End If
   ' add a new model to the collection
   Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & varFN, _
      mudtFam.ItemType)
   If udtM.IsFrozen Then
      nodN.Image = imSnowflake
   Else
      nodN.Image = imSun
   End If
   nodN.Expanded = True
   nodN.sorted = True
   Next intI

   ' enable attributes button
   cmdSetAttributes.Enabled = True

   ' force event to set active model
   treModels_Click

Cancel:

   UpdateTab0ControlStates

   Exit Sub

End Sub

Private Sub mnuFileImportItem_Click()

   Dim udtIni As New IniFile
   Dim strFN As String

   ' clear out everything
   ClearControls

   With cdlCD
      .InitDir = IN_DIRECTORY
      .FileName = ""
      .CancelError = True
      .DialogTitle = "Open locked item"
      .Filter = "Item Doc Files (*.doc)|*.doc|"
      .DefaultExt = ".doc"
      .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
      On Error GoTo Cancel
      .ShowOpen
      On Error GoTo 0
      strFN = .FileName
   End With
'   End If

   ' exit if there's no file name
   If Len(strFN) = 0 Then
      Exit Sub
   End If

   ' don't allow locked item to be opened if it's not in the "IN" directory
```

```
If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
   ' do nothing
Else
   Call MsgBox("Locked item must be located in " & IN_DIRECTORY, _
      vbExclamation, "Error")
   Exit Sub
End If

' set the FN of the ini that accompanies the locked item
udtIni.FN = IN_DIRECTORY & ExtractFileNameNoExt(strFN) & ".ini"

Dim udtW As New Win32API

If udtW.FileExists(udtIni.FN) = False Then
   Call MsgBox("Ini file must accompany locked item " & ExtractFileName(strFN) & _
      ".", vbExclamation, "Error")
   Exit Sub
End If

Dim udtProgram As Program
Dim udtDeliveryMode As DeliveryMode
Dim udtItemType As ItemType
Dim strAccNum As String

' find out about this locked item from the .ini file
Select Case udtIni.GetProfileString("LockedItemData", "Program")
   Case "GRE"
      udtProgram = prGRE
   Case "GMAT"
      udtProgram = prGMAT
   Case "SAT"
      udtProgram = prSAT
   Case "Not Found"
      Call MsgBox("No Program entry found in ini file " & ExtractFileName(strFN) & _
         ".", vbExclamation, "Error")
      Exit Sub
End Select

Select Case udtIni.GetProfileString("LockedItemData", "DeliveryMode")
   Case "CBT"
      udtDeliveryMode = dmCBT
   Case "PPT"
      udtDeliveryMode = dmPPT
   Case "Not Found"
      Call MsgBox("No DeliveryMode entry found in ini file " & ExtractFileName(strFN) & _
            ".", vbExclamation, "Error")
        Exit Sub
    End Select

    Select Case udtIni.GetProfileString("LockedItemData", "ItemType")
        Case "MC Item", "QantDisc", "MC", "Multiple Choice"
            udtItemType = ptStandardMC
        Case "DataSuff", "DS", "Data Sufficiency"
            udtItemType = ptDataSuff
        Case "QC Discrete", "QantComp", "QC", "Quantitative Comparison"
            udtItemType = ptQuantComp
        Case "Not Found"
            Call MsgBox("No ItemType entry found in ini file " & ExtractFileName(strFN) & _
                ".", vbExclamation, "Error")
            Exit Sub
    End Select

    strAccNum = udtIni.GetProfileString("LockedItemData", "LockedAccnum")
    If strAccNum = "Not Found" Then strAccNum = ""

    ' initialize locked item object
    Dim udtLI As New LockedItem

    udtLI.LockedItemFileName = strFN
    udtLI.WordInstance = mudtWord

    If udtLI.OpenLockedItemDoc = False Then ' we couldn't figure out what doc and item type it
was
        Call MsgBox("Locked item file appears to be damaged.", vbExclamation, "Error")
        udtLI.CloseLockedItemDoc
        Exit Sub
    End If

    With cdlCD
        .FileName = ""
        .DialogTitle = "Save new family based on this locked item as"
        .Filter = "Model Doc Files (*$R.doc)|*$R.doc|"
        .DefaultExt = ".doc"
        .Flags = cdlOFNHideReadOnly
        On Error GoTo CloseAndCancel
        .ShowSave
        On Error GoTo 0
        strFN = .FileName
    End With
'   End If
' see if an FN was entered
If Len(strFN) = 0 Then
   Beep
   Exit Sub
End If

strFN = UCase(strFN)

' check the extension
If (InStr(1, strFN, ".doc", vbTextCompare)) = 0 Then
   Call MsgBox("Invalid file name extension.", vbExclamation, "Error")
   Exit Sub
End If

Dim varI As Variant

' embed $R into FN if the user hasn't
If InStr(1, strFN, "$R.doc", vbTextCompare) = 0 Then
   varI = InStr(1, strFN, ".doc", vbTextCompare)
   strFN = Mid(strFN, 1, varI - 1) & "$R.doc"
End If

' check for unique FN
Dim udtWAPI As New Win32API

If udtWAPI.FileExists(strFN) Then
   Call MsgBox("File name " & _
      ExtractFileName(strFN) & " is not unique.", _
      vbExclamation, "Error")
   Exit Sub
End If

' copy the ini file of the locked item to the family name
Call FileCopy(udtIni.FN, left(strFN, Len(strFN) - 3) & "ini")

Dim strShortFN As String
strShortFN = ExtractFileName(strFN)

' create a new family object
Set mudtFam = New Family

' put family name on status bar
stbS.Panels(pnFamilyName) = strShortFN

' set file name, program, and item type
```

```
mudtFam.FileName = strFN
mudtFam.Program = udtProgram
mudtFam.ItemType = udtItemType
mudtFam.AccNum = strAccNum
mudtFam.IsDirty = True

' format tab 2
Call FormatTab2(mudtFam.ItemType)

' copy correct Word template to new model FN
Select Case mudtFam.ItemType

   Case ptStandardMC
      FileCopy App.path & "\TCASMC.doc", strFN

   Case ptQuantComp
      FileCopy App.path & "\TCAQC.doc", strFN

   Case ptDataSuff
      FileCopy App.path & "\TCADS.doc", strFN

End Select

Dim nodN As Node

' clear out the treeview box
treModels.Nodes.Clear

' add the new root
Set nodN = treModels.Nodes.Add(, , "R", strShortFN, imSun)
nodN.Expanded = True
nodN.sorted = True
nodN.Selected = True

Call mudtFam.Models.AddNew(strFN, mudtFam.ItemType)

mudtFam.Generic = False
mudtFam.Proximity = prNear

' enable attributes button
cmdSetAttributes.Enabled = True

' force event to set attributes
cmdSetAttributes_Click
```

```
   ' force event to set active model
   treModels_Click

   Select Case udtItemType
      Case ptStandardMC
         Select Case udtDeliveryMode
            Case dmCBT
               Call udtLI.ConvertCBTSMCItem
            Case dmPPT
               Call udtLI.ConvertPPTSMCItem
         End Select
      Case ptDataSuff
         Call udtLI.ConvertDSItem
      Case ptQuantComp
         Select Case udtDeliveryMode
            Case dmCBT
               Call udtLI.ConvertCBTQCItem
            Case dmPPT
               Call udtLI.ConvertPPTQCItem
         End Select
   End Select

CloseAndCancel:

   udtLI.CloseLockedItemDoc

Cancel:

   UpdateTab0ControlStates

   Exit Sub

End Sub

Private Sub mnuFileExit_Click()

   Call Form_Unload(0)
   End

End Sub

'Private Sub ReturnToTab0()
'
'   Dim intPrevTab As Integer
'
```

```
'   If sstMainTab.Tab = 0 Then Exit Sub
'
'   intPrevTab = sstMainTab.Tab
'   sstMainTab.Tab = 0
'   Call sstMainTab_Click(intPrevTab)
'
'End Sub
'
Private Sub mnuFilePrintSetup_Click()

    cdlCD.Flags = cdlPDPrintSetup

    On Error GoTo Cancel
    cdlCD.ShowPrinter
    On Error GoTo 0

Cancel:

    Exit Sub

End Sub

Private Sub mnuHelpAbout_Click()

    frmAbout.Show vbModal

End Sub

Private Sub mnuTreeExtend_Click()

    Dim nodN As Node
    Dim strFN As String
    Dim strNewFN As String
    Dim strKey As String
    Dim strT As String
    Dim strNewKey As String

    If treModels.SelectedItem Is Nothing Then Exit Sub

    Set nodN = treModels.SelectedItem
    strFN = nodN.Text

    ' confirm this operation
    If MsgBox("Make a child model from model " & strFN & "?", _
        vbQuestion + vbYesNo, "Confirm") = vbNo Then
    Exit Sub
  End If

  strKey = ModelKey(strFN)

  strNewKey = NextModelKey(strFN)

  ' add the child to the tree
  strNewFN = ModelEmbedKey(strFN, strNewKey)
  Set nodN = treModels.Nodes.Add(strKey, tvwChild, strNewKey, strNewFN)
  nodN.Expanded = True
  nodN.sorted = True
  nodN.Image = imSun

  ' deactivate active model to close it before file copies, if the active
  ' model is being extended.

  Dim blnReopenModel As Boolean

  blnReopenModel = False
  If strFN = stbS.Panels(pnActiveModelName) Then
    Call mudtFam.ActiveModel.CloseDoc
    blnReopenModel = True
  End If

  ' make a copy of the parent's word doc for this child
  Call FileCopy(IN_DIRECTORY & strFN, IN_DIRECTORY & strNewFN)

  ' make a copy of the parent's model file for this child
  Call FileCopy(IN_DIRECTORY & ModelFileName(strFN), IN_DIRECTORY &
ModelFileName(strNewFN))

  ' add the child's model to the model collection. "Thaw" the child.
  Dim udtM As Model
  Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & strNewFN, _
    mudtFam.ItemType)
  udtM.IsFrozen = False

  ' reset the clone index of the child
  udtM.LastClone = 0

  ' reset the checksums
  udtM.InitChecksums

  ' save it
```

```
    udtM.WriteModel

    If blnReopenModel Then
       Call mudtFam.ActiveModel.OpenDoc(mudtWord)
    End If

End Sub

Private Sub mnuTreeRemove_Click()

    Dim nodN As Node
    Dim strFN As String
    Dim strKey As String

    If treModels.SelectedItem Is Nothing Then Exit Sub

    Set nodN = treModels.SelectedItem
    strFN = nodN.Text

    strKey = ModelKey(strFN)

    Dim colIndices As New Collection

    ' don't remove if this node or any descendant nodes are frozen
    Dim udtModel As Model

    ' check selected node
    If treModels.SelectedItem.index = 1 Then ' it's the root model
       Call MsgBox("The root model can't be removed.", vbExclamation, "Error")
       Exit Sub
    End If

    Set udtModel = mudtFam.Models.Item(treModels.SelectedItem)
    If udtModel.IsFrozen Then
       Call MsgBox("Can't remove frozen model.", vbExclamation, "Error")
       Exit Sub
    Else
       Call colIndices.Add(treModels.SelectedItem.index)
    End If

    Dim blnDone As Boolean
    blnDone = False

    ' check if any of it's descendants are frozen
    Do
```

```
        Set nodN = nodN.Child
        If nodN Is Nothing Then
          ' do nothing
        Else
          Do
            If mudtFam.Models.Item(nodN.Text).IsFrozen Then
                Call MsgBox("Can't remove model with one or more frozen descendants.", _
                  vbExclamation, "Error")
                Exit Sub
            End If
            Call colIndices.Add(nodN.index)
          Loop Until nodN.index = nodN.LastSibling.index
        End If
      Loop Until nodN Is Nothing

      ' confirm this operation
      If MsgBox("Remove model " & strFN & " and it's children?", _
        vbQuestion + vbYesNo, "Confirm") = vbNo Then
        Exit Sub
      End If

      ' close active model document as we're deleting it
      mudtFam.ActiveModel.CloseDoc

      mudtFam.ActiveModel = Nothing
      stbS.Panels(pnActiveModelIcon).Picture = Nothing
      stbS.Panels(pnActiveModelName) = ""

      Dim varIndex As Variant

      ' remove all effected models from the family
      For Each varIndex In colIndices
        Call mudtFam.Models.Remove(treModels.Nodes(varIndex))
        Kill IN_DIRECTORY & left(treModels.Nodes(varIndex), _
          Len(treModels.Nodes(varIndex)) - 3) & "*"
      Next varIndex

      ' remove them from the tree control
      Call treModels.Nodes.Remove(colIndices(1))

    End Sub

    Private Sub mnuVariablesAdd_Click()

      frmVariable.AddEditFlag = aeAdd
```

```
End Sub

Private Sub mnuVariablesEdit_Click()

   frmVariable.AddEditFlag = aeEdit

End Sub

Private Sub mnuVariablesRemove_Click()

   Dim intInd As Integer

   intInd = lstVariables.ListIndex ' Get index

   ' Make sure list item is selected
   If intInd < 0 Then
      Beep
      Exit Sub
   End If

   Dim strVN As String

   strVN = mudtFam.ActiveModel.Variables.Item(Str(lstVariables.ItemData(intInd))).name

   ' confirm this operation
   If MsgBox("Remove variable " & strVN & "?", _
      vbQuestion + vbYesNo, "Confirm") = vbNo Then
      Exit Sub
   End If

   ' Remove the variable from the collection using the key in the list box
   Call mudtFam.ActiveModel.Variables.Remove(Str(lstVariables.ItemData(intInd)))

   ' Remove the variable from the list box
   Call lstVariables.RemoveItem(intInd)

   UpdateTab1ControlStates

End Sub

'Empty the variable list box
Private Sub mnuVariablesRemoveAll_Click()

   ' confirm this operation
   If MsgBox("Remove all variables?", _
        vbQuestion + vbYesNo, "Confirm") = vbNo Then
        Exit Sub
    End If

    'clear the list box
    lstVariables.Clear

    ' empty the collection
    mudtFam.ActiveModel.Variables.Clear

    UpdateTab1ControlStates

End Sub

Private Sub mnuVariablesEnableAll_Click()

    Call SetAllCheckboxes(True)

    UpdateTab1ControlStates

End Sub

Private Sub mnuVariablesDisableAll_Click()

    Call SetAllCheckboxes(False)

    UpdateTab1ControlStates

End Sub

Private Sub mnuVariablesTest_Click()

    Call TestConstraints(tcTestVariables)

End Sub
Private Sub mnuConstraintsAdd_Click()

    ' set the add flag for frmConstraints
    frmConstraints.AddEditFlag = aeAdd
    ' set the list box
    frmConstraints.ListBox = lstConstraints(mintConstrLBInd)
    ' set the model
    frmConstraints.Model = mudtFam.ActiveModel
    ' set the constraint type
    frmConstraints.ConstraintType = mintConstrLBInd
    ' crank up the form
    frmConstraints.Show vbModal

    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsEdit_Click()

    If lstConstraints(mintConstrLBInd).ListIndex >= 0 Then ' Make sure list item is selected
        ' set the edit flag for frmConstraints
        frmConstraints.AddEditFlag = aeEdit
        ' set the list box
        frmConstraints.ListBox = lstConstraints(mintConstrLBInd)
        ' set the model
        frmConstraints.Model = mudtFam.ActiveModel
        ' set the constraint
        With lstConstraints(mintConstrLBInd)
            frmConstraints.Constraint = _
                mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(.ListIndex)))
        End With
        ' set the constraint type
        frmConstraints.ConstraintType = mintConstrLBInd
        ' crank up the form
        frmConstraints.Show vbModal
    Else
        Beep
    End If

    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsRemove_Click()

    Dim intInd As Integer

    intInd = lstConstraints(mintConstrLBInd).ListIndex ' Get index

    ' Make sure list item is selected
    If intInd < 0 Then
        Beep
        Exit Sub
    End If
    Dim udtCon As Constraint
    Set udtCon = _
mudtFam.ActiveModel.Constraints.Item(Str(lstConstraints(mintConstrLBInd).ItemData(intInd))
)

    ' confirm this operation
    If MsgBox("Remove constraint " & udtCon.ConstraintString & "?", _
      vbQuestion + vbYesNo, "Confirm") = vbNo Then
      Exit Sub
    End If

    ' Remove the variable from the collection using the key in the list box
    Call
mudtFam.ActiveModel.Constraints.Remove(Str(lstConstraints(mintConstrLBInd).ItemData(intI
nd)))

    ' Remove the variable from the list box
    Call lstConstraints(mintConstrLBInd).RemoveItem(intInd)

    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsRemoveAll_Click()

      ' confirm this operation
    If MsgBox("Remove all constraints in this list box?", _
      vbQuestion + vbYesNo, "Confirm") = vbNo Then
       Exit Sub
    End If

    'clear the list box
    lstConstraints(mintConstrLBInd).Clear

    ' empty the collection
    Call mudtFam.ActiveModel.Constraints.Clear(mintConstrLBInd)

    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsEnableAll_Click()

    Call SetAllCheckboxes(True)
    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsDisableAll_Click()

    Call SetAllCheckboxes(False)

    Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsTest_Click()

    cmdSaveModel_Click ' force a save

    Select Case mintConstrLBInd
        Case ctVariation
            Call TestConstraints(tcTestVariationConstraints)
        Case ctDistractor
            Call TestConstraints(tcTestDistractorConstraints)
    End Select

End Sub

Private Sub mnuAcceptedProfile_Click()

    Dim udtClone As Clone
    Dim intI As Integer

    ' set the family
    frmDifficulty.Family = mudtFam

    ' set the clone
    With lstAccepted
        For intI = 0 To .ListCount - 1
            If .Selected(intI) Then
                Set udtClone = _
                    mudtFam.Clones.Item(Str(.ItemData(intI)))
                frmDifficulty.Clone = udtClone
                Exit For
            End If
        Next intI
    End With
    ' give frmDifficulty a caption
    frmDifficulty.Caption = "Profile of variant " & _
      ExtractFileName(udtClone.FileName)

    ' crank up the form
    frmDifficulty.Show vbModal

    If udtClone.IsRouted Then
      lstAccepted.List(intI) = udtClone.FileName & ": Routed to TCS"
    Else
      lstAccepted.List(intI) = udtClone.FileName
    End If

End Sub

Private Sub mnuAcceptedCopy_Click()

    Dim udtClone As Clone

    ' this menu option is only active if a variant with a completed profile
    ' is currently selected.
    With lstAccepted
      Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
    End With

    ' copy necessary stuff into a holding area
    Set mudtClone = udtClone

    UpdateTab0ControlStates

End Sub

' this menu option is only active if a profile has been copied
Private Sub mnuAcceptedPaste_Click()

    Dim udtClone As Clone
    Dim intI As Integer

    With lstAccepted
      If .SelCount > 0 Then
        ' confirm this operation
        If MsgBox("Paste profile of variant " & mudtClone.FileName & _
          " to all selected variants?", _
          vbQuestion + vbYesNo, "Confirm") = vbNo Then
          Exit Sub
```

```
            End If
            For intI = 0 To .ListCount - 1
               If .Selected(intI) Then
                  Set udtClone = mudtFam.Clones.Item(Str(.ItemData(intI)))
                  ' copy necessary stuff from the holding area
                  udtClone.Domain = mudtClone.Domain
                  udtClone.BatchID = mudtClone.BatchID
                  udtClone.DeliveryMode = mudtClone.DeliveryMode
                  udtClone.Nature = mudtClone.Nature
                  udtClone.IsRouted = mudtClone.IsRouted
                  udtClone.TDEstimate = mudtClone.TDEstimate
                  udtClone.IsDifficultyCalculated = mudtClone.IsDifficultyCalculated
                  If udtClone.IsDifficultyCalculated Then
                     udtClone.DiffEst = mudtClone.DiffEst.Copy
                  End If
                  If udtClone.IsRouted Then
                     .List(intI) = udtClone.FileName & ": Routed to TCS"
                  Else
                     .List(intI) = udtClone.FileName
                  End If
               End If
            Next intI
         End If
      End With

   End Sub

   ' checks/unchecks all checkboxes in a listbox and enable/disable their
   ' associated variable or constraint objects

   Private Sub SetAllCheckboxes(ByVal blnBool As Boolean)

      Dim i As Integer

      For i = 0 To (mlstCurrentListBox.ListCount - 1)
         mlstCurrentListBox.Selected(i) = blnBool
      Next i

      Dim udtV As Variable
      Dim udtC As Constraint

      If mlstCurrentListBox.name = "lstVariables" Then
         For Each udtV In mudtFam.ActiveModel.Variables
            udtV.Enabled = blnBool
         Next udtV
```

```
    Else
        For i = 0 To (mlstCurrentListBox.ListCount - 1)
            Set udtC =
mudtFam.ActiveModel.Constraints.Item(Str(mlstCurrentListBox.ItemData(i)))
            udtC.Enabled = blnBool
        Next i
    End If

End Sub

Private Sub mwudtModelTest_PrologFinished()

End Sub

Private Sub sstMainTab_Click(PreviousTab As Integer)

    Static blnRecursing As Boolean
    Static bytMessage As Byte

    If blnRecursing Then
        Select Case bytMessage
            Case 1
                Call MsgBox("Open a model family using the File menu.", _
                    vbExclamation, "Error")
            Case 2
                Call MsgBox("Set the active model by clicking on a model.", _
                    vbExclamation, "Error")
        End Select
        blnRecursing = False
        Exit Sub
    End If

    ' error conditions
    If sstMainTab.Tab > 0 Then
        If treModels.Nodes.Count = 0 Then ' family hasn't been set
            bytMessage = 1
            blnRecursing = True
            sstMainTab.Tab = PreviousTab ' will trigger recursion
            Exit Sub
        End If
    End If

    If sstMainTab.Tab = 1 Or sstMainTab.Tab = 2 Then
        If mudtFam.ActiveModel Is Nothing Then ' active model has not been set
            bytMessage = 2
            blnRecursing = True
            sstMainTab.Tab = PreviousTab ' will trigger recursion
            Exit Sub
         End If
      End If

      ' if we got here, everything's ok!
      If PreviousTab = 2 Then
         txtNum2Generate = ""
      End If

      If PreviousTab = 1 Then
         If mudtFam.ActiveModel.IsDirty Then
            KillVariants 'delete any variants on tab 3
            mudtFam.ActiveModel.InitTempChecksums ' initialize temp checksums
         End If
      End If

      ' save family
      mudtFam.WriteFamily

      ' save the active model
      If mudtFam.ActiveModel Is Nothing Then
         ' do nothing
      Else
         mudtFam.ActiveModel.WriteModel
      End If

      Select Case sstMainTab.Tab

         Case 0
            ' enable new/open
            cmdSetAttributes.Default = True
            mnuFileNew.Enabled = True
            mnuFileOpen.Enabled = True
            mnuFileImportItem.Enabled = True
            If PreviousTab = 2 Then
               mudtFam.ActiveModel.CloseAllCloneDocs
               Call mudtFam.ActiveModel.OpenDoc(mudtWord)
            End If

            ' if there are no variants, disable the print button
            If lstAccepted.ListCount > 0 Then
               cmdPrintBatch.Enabled = True
            Else
```

```
        cmdPrintBatch.Enabled = False
      End If

    Case 1
      cmdSaveModel.Default = True
      ' disable new/open
      mnuFileNew.Enabled = False
      mnuFileOpen.Enabled = False
      mnuFileImportItem.Enabled = False
      ' warn if variants exist in lstDisposition and model isn't frozen
      If mudtFam.ActiveModel.IsFrozen = False Then
        If lstDisposition.ListCount > 0 Then ' variants exist
          Call MsgBox("Variants on tab 3 will be deleted if " & _
            "the model is changed.", vbInformation, "Warning")
        End If
      End If
      If PreviousTab = 0 Then
        mudtFam.CloseAllCloneDocs
        Call mudtFam.ActiveModel.OpenDoc(mudtWord)
      End If
      If PreviousTab = 2 Then
        mudtFam.ActiveModel.CloseAllCloneDocs
        Call mudtFam.ActiveModel.OpenDoc(mudtWord)
      End If

    Case 2
      cmdGenerate.Default = True
      ' disable new/open
      mnuFileNew.Enabled = False
      mnuFileOpen.Enabled = False
      mnuFileImportItem.Enabled = False

      ' disable the generate button
       cmdGenerate.Enabled = False

      ' if there are no variants, disable the print button
      If lstDisposition.ListCount > 0 Then
        cmdPrintVariants.Enabled = True
      Else
        cmdPrintVariants.Enabled = False
      End If

      If PreviousTab = 0 Then
        mudtFam.CloseAllCloneDocs
      End If
```

```
            ' display the currently selected document
            With lstDisposition
               If .ListCount > 0 Then ' a valid selection has been made
                  Call mudtFam.ActiveModel.Clones.Item _
                     (Str(.ItemData(.ListIndex))).OpenDoc(mudtWord, IN_DIRECTORY)
               Else
                  Call mudtFam.ActiveModel.OpenDoc(mudtWord)
               End If
            End With

      End Select

   End Sub

   ' restore full window drag, if necessary
   Private Sub sstMainTab_MouseMove(Button As Integer, _
      Shift As Integer, X As Single, Y As Single)

      Dim udtW As Win32API

      If mblnRestoreFullWindowDrag Then
         Set udtW = New Win32API
         udtW.TurnOnFullWindowDrag
         mblnRestoreFullWindowDrag = False
      End If

      If mudtWord Is Nothing Then Exit Sub

      If sstMainTab.Tab = 1 Then ' do this first, as there will be an active doc
                             ' on tab 1
         If mudtWord.WordApp.ActiveDocument.Saved = False And _
            cmdSaveModel.Enabled = False Then
            If Not mudtFam.ActiveModel.IsFrozen Then
               mudtFam.ActiveModel.IsDirty = True
               UpdateTab1ControlStates
            End If
         End If
      End If

   End Sub

   Private Sub treModels_Click()

      Dim nodN As Node
```

```
If treModels.SelectedItem Is Nothing Then Exit Sub

Set nodN = treModels.SelectedItem

' put model icon and name on status bar
stbS.Panels(pnActiveModelIcon).Picture = imlI.ListImages(nodN.Image).Picture
stbS.Panels(pnActiveModelName) = treModels.SelectedItem

' close doc for existing active model
If mudtFam.ActiveModel Is Nothing Then
   ' do nothing
Else
   mudtFam.ActiveModel.CloseDoc
End If

' set the new active model and activate it
mudtFam.ActiveModel = mudtFam.Models.Item(treModels.SelectedItem)
Call mudtFam.ActiveModel.OpenDoc(mudtWord)

' clear out the Variable list box
lstVariables.Clear

' populate the variable list box with this model's variables
Dim udtVar As Variable

For Each udtVar In mudtFam.ActiveModel.Variables
   With lstVariables
      Call .AddItem(udtVar.ScreenFormat)
      .ItemData(.ListCount - 1) = udtVar.index
      .Selected(.ListCount - 1) = udtVar.Enabled
   End With
Next udtVar

Dim intI

' clear out the constraint list boxes
lstConstraints(0).Clear
lstConstraints(1).Clear

' populate the constraint list boxes with this model's constraints
Dim udtCon As Constraint

For Each udtCon In mudtFam.ActiveModel.Constraints
   intI = udtCon.ConstraintType
   With lstConstraints(intI)
```

```
            Call .AddItem(udtCon.ConstraintString)
            .ItemData(.ListCount - 1) = udtCon.index
            .Selected(.ListCount - 1) = udtCon.Enabled
         End With
      Next udtCon

      ' populate comments form
      frmComments.Comment = mudtFam.ActiveModel.Comments

      ' clear out the clone disposition list box
      lstDisposition.Clear

      ' populate the clone list box with this model's clones
      Dim udtClone As Clone

      With lstDisposition
         For Each udtClone In mudtFam.ActiveModel.Clones
            Call .AddItem(ExtractFileName(udtClone.FileName))
            .ItemData(.ListCount - 1) = udtClone.index
         Next udtClone
      End With

      ' save the active model
      mudtFam.ActiveModel.WriteModel

      ' adjust menu/button states depending on active model properties
      UpdateTab1ControlStates
      UpdateTab2ControlStates

      ' enable extend
      mnuTreeExtend.Enabled = True

   End Sub

   Private Sub treModels_MouseUp(Button As Integer, Shift As Integer, _
      X As Single, Y As Single)

      If treModels.Nodes.Count > 0 Then
         If Button = vbRightButton Then
            PopupMenu mnuTree
         End If
      End If

   End Sub
```

```
Private Sub txtNum2Generate_Change()

'   If Val(txtNum2Generate) > 0 Then
'       cmdGenerate.Enabled = True
'   Else
'       cmdGenerate.Enabled = False
'   End If

End Sub

Private Sub txtVariablize_GotFocus()

    If mudtWord.DocumentsCount = 0 Then
        Beep
    Else
        If mudtWord.SelectionType < wdSelectionNormal Then
            Call MsgBox("Nothing is selected.", vbExclamation, "Error")
        Else
            Call AddUndefinedVariables(mudtWord.SelectionText)
        End If
    End If

End Sub

' scans a string for undefined variable names and add them to
' the variable collection and list box

Public Sub AddUndefinedVariables(ByVal strNames As String)

    Dim colC As Collection
    Dim strS As Variant
    Dim udtVar As Variable
    Dim colDummy As New Collection

    Set colC = UndefinedNames(strNames)

    ' don't do it if the model is frozen!
    If Not mudtFam Is Nothing Then
        If Not mudtFam.ActiveModel Is Nothing Then
            If mudtFam.ActiveModel.IsFrozen Then
                Call MsgBox("Variables cannot be added to a frozen model.", _
                    vbExclamation, "Error")
                Exit Sub
            End If
        End If
End If

For Each strS In colC

   If MsgBox("Auto-define variable " & strS & "?", vbQuestion + vbYesNo, _
      "New variable detected") = vbYes Then

      Select Case left(strS, 1)
         Case "I"
            Set udtVar = mudtFam.ActiveModel.Variables.AddInteger(strS, _
               True, "1", "100", "1", False, True)
         Case "R"
            Set udtVar = mudtFam.ActiveModel.Variables.AddReal(strS, _
               True, "1", "100", "1", False, True, True, ".01", True)
         Case "S"
            Set udtVar = mudtFam.ActiveModel.Variables.AddString(strS, _
               True, True, Chr(164), True, colDummy)
         Case "F"
            Set udtVar = mudtFam.ActiveModel.Variables.AddFraction(strS, _
               True, "1", "1", "100", "1", "1", ",1", False, True, False)
         Case "U"
            Set udtVar = mudtFam.ActiveModel.Variables.AddUntyped(strS, _
               True, False)
         Case Else ' assume untyped
            Set udtVar = mudtFam.ActiveModel.Variables.AddUntyped(strS, _
               True, False)
      End Select

      With lstVariables
         ' Add the new variable to the variable list box
         Call .AddItem(udtVar.ScreenFormat)
         ' Set ItemData to index value of the variable object
         .ItemData(.ListCount - 1) = udtVar.index
         ' Check the check box
         .Selected(.ListCount - 1) = True
      End With

   End If

Next strS

' update control states
If colC.Count > 0 Then
   UpdateTab1ControlStates
End If
End Sub

' accepts a string and parses it for undefined variable names.  Returns a
' collection of the variable names that are unique.

Public Function UndefinedNames(ByVal strS As String) As Collection

  Dim lngStart As Long
  Dim lngEnd As Long
  Dim strT As String
  Dim byt1 As Byte
  Dim byt2 As Byte
  Dim colC As New Collection
  Dim blnDup As Boolean
  Dim varT As Variant

  ' parse the variable names out of strS
  For lngStart = 1 To Len(strS)
    byt1 = Asc(Mid(strS, lngStart, 1))
    If byt1 >= 65 And byt1 <= 90 Then
      For lngEnd = lngStart + 1 To Len(strS)
        byt2 = Asc(Mid(strS, lngEnd, 1))
        Select Case byt2
          Case 48 To 57, 65 To 90, 97 To 122
            '  if it's 0 to 9, A to Z, or a to z, continue searching
          Case Else
            ' if it's not, assume end of variable name has been found
            Exit For
        End Select
      Next lngEnd
      strT = Mid(strS, lngStart, lngEnd - lngStart)
      ' throw name away if it's already in colC
      blnDup = False
      For Each varT In colC
        If UCase(varT) = UCase(strT) Then
          blnDup = True
        End If
      Next varT
      ' make sure name is not a Prolog function
      If blnDup = False Then
        ' throw name away if it's already in the main variable collection
        If mudtFam.ActiveModel.Variables.UniqueName(strT) Then
          Call colC.Add(strT)
        End If
      End If
```

```
            lngStart = lngEnd
        End If
    Next lngStart

    Set UndefinedNames = colC

End Function

Private Sub TestConstraints(ByVal udtTestType As TestType)

    Dim strVN As String
    Dim blnUnderconstrained As Boolean
    Dim blnTestAborted As Boolean

    If mudtFam.ActiveModel.ConstraintsOK(udtTestType, mudtProlog, _
        blnUnderconstrained, blnTestAborted, strVN) Then
            Call MsgBox("Looks good!", vbExclamation, "Test Result")
    ElseIf blnTestAborted Then
        Call MsgBox("Test aborted!", vbExclamation, "Test Result")
    ElseIf blnUnderconstrained Then
        Call MsgBox("Variable " & strVN & " is underconstrained!", _
            vbExclamation, "Test Result")
    Else
        Call MsgBox("No solutions exist!", vbExclamation, "Test Result")
    End If

End Sub

' displays the family attributes on the status bar

Private Sub UpdateFamilyAttributes()

    Select Case mudtFam.Program
        Case prGRE
            stbS.Panels(pnProgramName) = "GRE"
        Case prGMAT
            stbS.Panels(pnProgramName) = "GMAT"
        Case prSAT
            stbS.Panels(pnProgramName) = "SAT"
    End Select

    Select Case mudtFam.ItemType
        Case ptStandardMC
            stbS.Panels(pnItemType) = "SMC"
        Case ptQuantComp
```

```
            stbS.Panels(pnItemType) = "QC"
        Case ptDataSuff
            stbS.Panels(pnItemType) = "DS"
    End Select

    If mudtFam.Generic Then
        stbS.Panels(pnGeneric) = "Generic"
    Else
        stbS.Panels(pnGeneric) = "Non generic"
    End If

    Select Case mudtFam.Proximity
        Case prNear
            stbS.Panels(pnProximity) = "Near"
        Case prMedium
            stbS.Panels(pnProximity) = "Medium"
        Case prFar
            stbS.Panels(pnProximity) = "Far"
    End Select

End Sub

' returns the model file name given the doc file name

Private Function ModelFileName(ByVal strDocFN As String) As String

    ModelFileName = left(strDocFN, Len(strDocFN) - 4) & ".mdl"

End Function

' extracts the key from a model file name
Private Function ModelKey(ByVal strFN As String) As String

    Dim varI1 As Variant
    Dim varI2 As Variant
    Dim intI As Integer
    Dim strS As String

    varI1 = InStr(1, strFN, "$")
    varI2 = InStr(varI1, strFN, ".")

    ' strip off numbers or spaces to the left of the "."
    intI = varI2
    Do While intI > varI1
        intI = intI - 1
        strS = Mid(strFN, intI, 1)
        If Asc(strS) >= 65 And Asc(strS) <= 91 Then ' it's A to Z
            varI2 = intI + 1
            Exit Do
        End If
    Loop

    ModelKey = Mid(strFN, varI1 + 1, varI2 - varI1 - 1)

End Function

' embeds a new key into a model file name
Private Function ModelEmbedKey(ByVal strFN As String, ByVal strNewKey As String) _
    As String

    Dim varI1 As Variant
    Dim varI2 As Variant
    Dim intI As Integer
    Dim strS As String

    varI1 = InStr(1, strFN, "$")
    varI2 = InStr(varI1, strFN, ".")

    ' strip off numbers or spaces to the left of the "."
    intI = varI2
    Do While intI > varI1
        intI = intI - 1
        strS = Mid(strFN, intI, 1)
        If Asc(strS) >= 65 And Asc(strS) <= 91 Then ' it's A to Z
            varI2 = intI + 1
            Exit Do
        End If
    Loop

    ModelEmbedKey = left(strFN, varI1) & strNewKey & right(strFN, 4)

End Function

' returns the key of the next child for this model
Private Function NextModelKey(strFN As String) As String

    Dim nodN As Node
    Dim strNewFN As String
    Dim strIndex As String
    Dim strT As String
    strIndex = ModelKey(strFN)


    Dim intI As Integer

    ' when the key can't be found in the Nodes collection, an error
    ' is raised.  When the error is raised, the first available letter
    ' of the alphabet has been found.

    On Error GoTo Found

    For intI = 65 To 90 ' A thru Z
      strT = Chr(intI)
      Set nodN = treModels.Nodes.Item(strIndex & strT)
    Next intI

    On Error GoTo 0

    Call MsgBox("Can't add another child model to this parent", _
      vbExclamation, "Error")
    Exit Function

Found:

    NextModelKey = strIndex & strT
    Exit Function

End Function

' resets controls and variables when a new family is opened.
Private Sub ClearControls()

    If mudtFam Is Nothing Then
      ' do nothing
    Else
      mudtFam.WriteFamily
        If mudtFam.ActiveModel Is Nothing Then
          ' do nothing
        Else
          mudtFam.ActiveModel.WriteModel
        End If
    End If

    mudtWord.CloseAllDocs
    Set mudtFam = Nothing
    Set mudtClone = Nothing

    treModels.Nodes.Clear
    lstVariables.Clear
    lstDisposition.Clear
    lstAccepted.Clear
    stbS.Panels(pnProgramName) = ""
    stbS.Panels(pnFamilyName) = ""
    stbS.Panels(pnItemType) = ""
    stbS.Panels(pnGeneric) = ""
    stbS.Panels(pnProximity) = ""
    stbS.Panels(pnActiveModelIcon).Picture = Nothing
    stbS.Panels(pnActiveModelName) = ""
    frmComments.Comment = ""
    mnuAcceptedCopy.Enabled = False
    mnuAcceptedPaste.Enabled = False

End Sub

' used to reformat tab 2 as QC and DS don't need a distractor listbox
Private Sub FormatTab2(ByVal udtItemType As ItemType)

    Select Case udtItemType
        Case ptStandardMC
            ' turn on the distractor list box
            lblDistractor.Visible = True
            lstConstraints(1).Visible = True
            cmdConstraintAdd(1).Visible = True
            cmdConstraintEdit(1).Visible = True
            cmdConstraintRemove(1).Visible = True
            cmdConstraintTest(1).Visible = True
        Case ptQuantComp
            ' turn off the distractor list box
            lblDistractor.Visible = False
            lstConstraints(1).Visible = False
            cmdConstraintAdd(1).Visible = False
            cmdConstraintEdit(1).Visible = False
            cmdConstraintRemove(1).Visible = False
            cmdConstraintTest(1).Visible = False
        Case ptDataSuff
            ' turn off the distractor list box
            lblDistractor.Visible = False
            lstConstraints(1).Visible = False
            cmdConstraintAdd(1).Visible = False
            cmdConstraintEdit(1).Visible = False
            cmdConstraintRemove(1).Visible = False
            cmdConstraintTest(1).Visible = False
    End Select

End Sub

' this method gets rid of all variants in the lstDisposition listbox,
' deletes them from disk, and removes them from the active model.

Private Sub KillVariants()

    Dim udtClone As Clone
    Dim intI As Integer

    With lstDisposition
        For intI = 0 To .ListCount - 1
            ' get object from active model's clone collection
            Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
            ' close the document
            udtClone.CloseDoc
            ' delete the clone file
            Kill IN_DIRECTORY & udtClone.FileName
            ' remove the clone from the active model's collection
            Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
        Next intI
        For intI = .ListCount - 1 To 0 Step -1
            ' remove the entry from the disposition list box
            Call .RemoveItem(intI)
        Next intI
    End With

End Sub

Private Sub UpdateTab0ControlStates()

    ' update model tree menu states
    With treModels
        If .Nodes.Count > 0 Then
            mnuTreeExtend.Enabled = True
            mnuTreeRemove.Enabled = True
            cmdTreeExtend.Enabled = True
            cmdTreeRemove.Enabled = True
        Else
            mnuTreeExtend.Enabled = False
            mnuTreeRemove.Enabled = False
            cmdTreeExtend.Enabled = False
            cmdTreeRemove.Enabled = False
        End If
    End With

    ' update accepted list box menu states
    With lstAccepted
        If .ListCount > 0 Then
            cmdPrintBatch.Enabled = True
            If .SelCount = 1 Then ' 1 item is selected
                mnuAcceptedProfile.Enabled = True
                mnuAcceptedCopy.Enabled = True
                cmdAcceptedEdit.Enabled = True
                cmdAcceptedCopy.Enabled = True
            ElseIf .SelCount > 1 Then ' more than one is selected
                mnuAcceptedProfile.Enabled = False
                mnuAcceptedCopy.Enabled = False
                cmdAcceptedEdit.Enabled = False
                cmdAcceptedCopy.Enabled = False
            End If
        Else ' nothings in the list box
            cmdPrintBatch.Enabled = False
            mnuAcceptedProfile.Enabled = False
            mnuAcceptedCopy.Enabled = False
            mnuAcceptedPaste.Enabled = False
            cmdAcceptedEdit.Enabled = False
            cmdAcceptedCopy.Enabled = False
            cmdAcceptedPaste.Enabled = False
        End If
    End With

    If mudtClone Is Nothing Then ' nothing to paste
        mnuAcceptedPaste.Enabled = False
        cmdAcceptedPaste.Enabled = False
    ElseIf lstAccepted.SelCount > 0 Then ' one or more are selected
        mnuAcceptedPaste.Enabled = True
        cmdAcceptedPaste.Enabled = True
    Else ' none are selected
        mnuAcceptedPaste.Enabled = False
        cmdAcceptedPaste.Enabled = False
    End If

    If mudtFam Is Nothing Then
        cmdDone.Enabled = False
    Else
       cmdDone.Enabled = True
    End If

End Sub

Private Sub UpdateTab1ControlStates(Optional ByVal intIndex As Integer = 0)

    Dim strCaption As String

    If mudtFam.ActiveModel.IsFrozen Then
       strCaption = "Browse"
    Else
       strCaption = "Edit"
    End If

    mnuVariablesEdit.Caption = strCaption
    cmdVariableEdit.Caption = strCaption
    mnuConstraintsEdit.Caption = strCaption
    cmdConstraintEdit(0).Caption = strCaption
    cmdConstraintEdit(1).Caption = strCaption

    ' update variable list box menu states
    If mudtFam.ActiveModel.IsFrozen Then
       mnuVariablesAdd.Enabled = False
       mnuVariablesEdit.Enabled = True
       mnuVariablesEnableAll.Enabled = False
       mnuVariablesDisableAll.Enabled = False
       mnuVariablesRemove.Enabled = False
       mnuVariablesRemoveAll.Enabled = False
       cmdVariableAdd.Enabled = False
       cmdVariableEdit.Enabled = True
       cmdVariableRemove.Enabled = False
    ElseIf lstVariables.ListCount > 0 Then
       mnuVariablesAdd.Enabled = True
       mnuVariablesEdit.Enabled = True
       mnuVariablesEnableAll.Enabled = True
       mnuVariablesDisableAll.Enabled = True
       mnuVariablesRemove.Enabled = True
       mnuVariablesRemoveAll.Enabled = True
       cmdVariableAdd.Enabled = True
       cmdVariableEdit.Enabled = True
       cmdVariableRemove.Enabled = True
    Else
       mnuVariablesAdd.Enabled = True
    mnuVariablesEdit.Enabled = False
    mnuVariablesEnableAll.Enabled = False
    mnuVariablesDisableAll.Enabled = False
    mnuVariablesRemove.Enabled = False
    mnuVariablesRemoveAll.Enabled = False
    cmdVariableAdd.Enabled = True
    cmdVariableEdit.Enabled = False
    cmdVariableRemove.Enabled = False
End If

' isfrozen should not effect state of test option
If lstVariables.ListCount > 0 Then
    mnuVariablesTest.Enabled = True
    cmdVariableTest.Enabled = True
Else
    mnuVariablesTest.Enabled = False
    cmdVariableTest.Enabled = False
End If

' update constraints list box menu states
If mudtFam.ActiveModel.IsFrozen Then
    mnuConstraintsAdd.Enabled = False
    mnuConstraintsEdit.Enabled = True
    mnuConstraintsEnableAll.Enabled = False
    mnuConstraintsDisableAll.Enabled = False
    mnuConstraintsRemove.Enabled = False
    mnuConstraintsRemoveAll.Enabled = False
    cmdConstraintAdd(0).Enabled = False
    cmdConstraintAdd(1).Enabled = False
    cmdConstraintEdit(0).Enabled = True
    cmdConstraintEdit(1).Enabled = True
    cmdConstraintRemove(0).Enabled = False
    cmdConstraintRemove(1).Enabled = False
ElseIf lstConstraints(intIndex).ListCount > 0 Then
    mnuConstraintsAdd.Enabled = True
    mnuConstraintsEdit.Enabled = True
    mnuConstraintsEnableAll.Enabled = True
    mnuConstraintsDisableAll.Enabled = True
    mnuConstraintsRemove.Enabled = True
    mnuConstraintsRemoveAll.Enabled = True
    cmdConstraintAdd(intIndex).Enabled = True
    cmdConstraintEdit(intIndex).Enabled = True
    cmdConstraintRemove(intIndex).Enabled = True
Else
    mnuConstraintsAdd.Enabled = True
    mnuConstraintsEdit.Enabled = False
    mnuConstraintsEnableAll.Enabled = False
    mnuConstraintsDisableAll.Enabled = False
    mnuConstraintsRemove.Enabled = False
    mnuConstraintsRemoveAll.Enabled = False
    cmdConstraintAdd(intIndex).Enabled = True
    cmdConstraintEdit(intIndex).Enabled = False
    cmdConstraintRemove(intIndex).Enabled = False
End If

' isfrozen should not effect state of test option
If lstConstraints(intIndex).ListCount > 0 Then
    mnuConstraintsTest.Enabled = True
    cmdConstraintTest(intIndex).Enabled = True
Else
    mnuConstraintsTest.Enabled = False
    cmdConstraintTest(intIndex).Enabled = False
End If

' flip the index
If intIndex = 0 Then
    intIndex = 1
Else
    intIndex = 0
End If

' update button states for the other constraint list box
If mudtFam.ActiveModel.IsFrozen = False Then
    If lstConstraints(intIndex).ListCount > 0 Then
        cmdConstraintAdd(intIndex).Enabled = True
        cmdConstraintEdit(intIndex).Enabled = True
        cmdConstraintRemove(intIndex).Enabled = True
    Else
        cmdConstraintAdd(intIndex).Enabled = True
        cmdConstraintEdit(intIndex).Enabled = False
        cmdConstraintRemove(intIndex).Enabled = False
    End If
End If

' isfrozen should not effect state of test option
If lstConstraints(intIndex).ListCount > 0 Then
    cmdConstraintTest(intIndex).Enabled = True
Else
    cmdConstraintTest(intIndex).Enabled = False
End If
```

```
    ' update import button
    If mudtFam.ActiveModel.IsFrozen Then
      cmdImportConstraints.Enabled = False
    Else
      cmdImportConstraints.Enabled = True
    End If

    ' if model frozen, disable save
    If mudtFam.ActiveModel.IsFrozen Then
      cmdSaveModel.Enabled = False
    Else
      If mudtFam.ActiveModel.IsDirty Then
        cmdSaveModel.Enabled = True
      Else
        cmdSaveModel.Enabled = False
      End If
    End If

  End Sub

  Private Sub UpdateTab2ControlStates()

    ' update disposition list box menu states
    If lstDisposition.ListCount > 0 And cmdGenerate.Caption = "Generate" Then
      mnuDispAccept.Enabled = True
      mnuDispDefer.Enabled = True
      mnuDispDiscard.Enabled = True
      mnuDispMakeModel.Enabled = True
      cmdPrintVariants.Enabled = True
      cmdPrintVariants.Enabled = True
      cmdDispAccept.Enabled = True
      cmdDispDefer.Enabled = True
      cmdDispDiscard.Enabled = True
      cmdDispMakeModel.Enabled = True
    Else
      mnuDispAccept.Enabled = False
      mnuDispDefer.Enabled = False
      mnuDispDiscard.Enabled = False
      mnuDispMakeModel.Enabled = False
      cmdPrintVariants.Enabled = False
      cmdPrintVariants.Enabled = False
      cmdDispAccept.Enabled = False
      cmdDispDefer.Enabled = False
      cmdDispDiscard.Enabled = False
      cmdDispMakeModel.Enabled = False
```

```
        End If
    End Sub
```

```
' Variable.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
Begin VB.Form frmVariable
  BorderStyle    =  4  'Fixed ToolWindow
  Caption        =  "Create or Change Variable"
  ClientHeight   =  4230
  ClientLeft     =  45
  ClientTop      =  285
  ClientWidth    =  6525
  LinkTopic      =  "Form1"
  MaxButton      =  0  'False
  MinButton      =  0  'False
  ScaleHeight    =  4230
  ScaleWidth     =  6525
  ShowInTaskbar  =  0  'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.ComboBox cboVarType
    Height         =  315
    ItemData       =  "Variable.frx":0000
    Left           =  2040
    List           =  "Variable.frx":0013
    Style          =  2  'Dropdown List
    TabIndex       =  1
    ToolTipText    =  "Select the variable type."
    Top            =  360
    Width          =  1695
  End
  Begin VB.CheckBox chkChecksum
    Caption        =  "Add to checksum"
    Height         =  375
    Left           =  240
    TabIndex       =  2
    ToolTipText    =  "Check this box to add this variable to the checksum calcuation."
    Top            =  840
    Value          =  1  'Checked
    Width          =  1815
  End
  Begin MSComDlg.CommonDialog cdlCD
    Left           =  5280
    Top            =  2520
    _ExtentX       =  847
    _ExtentY       =  847
  _Version     =  393216
End
Begin VB.CommandButton cmdVarExport
  Caption      =  "Export Strings"
  Height       =  495
  Left         =  5160
  TabIndex     =  7
  ToolTipText  =  "Click here to export a set of strings."
  Top          =  1920
  Width        =  1215
End
Begin VB.CommandButton cmdVarImport
  Caption      =  "Import Strings"
  Height       =  495
  Left         =  5160
  TabIndex     =  6
  ToolTipText  =  "Click here to import a set of strings."
  Top          =  1320
  Width        =  1215
End
Begin VB.TextBox txtVariableName
  Height       =  315
  Left         =  240
  TabIndex     =  0
  ToolTipText  =  "Enter the variable name here."
  Top          =  360
  Width        =  1695
End
Begin VB.CommandButton cmdVarCancel
  Caption      =  "Cancel"
  Height       =  495
  Left         =  5160
  TabIndex     =  5
  ToolTipText  =  "Click here to return without saving changes."
  Top          =  720
  Width        =  1215
End
Begin VB.CommandButton cmdVarOK
  Caption      =  "OK"
  Default      =  -1  'True
  Height       =  495
  Left         =  5160
  TabIndex     =  4
  ToolTipText  =  "Click here to save changes and return."
  Top          =  120
```

```
    Width         =  1215
  End
  Begin ComctlLib.ListView lvwTemp
    Height        =  375
    Left        =  5280
    TabIndex      =  43
    Top         =  3120
    Visible       =  0  'False
    Width         =  495
    _ExtentX      =  873
    _ExtentY      =  661
    View          =  3
    Arrange       =  2
    LabelEdit     =  1
    MultiSelect   =  -1  'True
    LabelWrap     =  -1  'True
    HideSelection  =  -1  'True
    _Version      =  327682
    ForeColor     =  -2147483640
    BackColor     =  -2147483643
    BorderStyle   =  1
    Appearance    =  1
    NumItems      =  0
  End
  Begin ComctlLib.ListView lvwDummy
    Height        =  375
    Left        =  5280
    TabIndex      =  44
    Top         =  3600
    Visible       =  0  'False
    Width         =  495
    _ExtentX      =  873
    _ExtentY      =  661
    View          =  3
    Arrange       =  2
    LabelEdit     =  1
    MultiSelect   =  -1  'True
    LabelWrap     =  -1  'True
    HideSelection  =  -1  'True
    _Version      =  327682
    ForeColor     =  -2147483640
    BackColor     =  -2147483643
    BorderStyle   =  1
    Appearance    =  1
    NumItems      =  0
```

```
End
Begin VB.Frame fraString
  BorderStyle    =  0  'None
  Height       =  2895
  Left        =  240
  TabIndex      =  9
  Top         =  1200
  Width        =  4815
  Begin ComctlLib.ListView lvwStrings
    Height        =  1815
    Left         =  0
    TabIndex       =  42
    Top          =  720
    Width         =  3975
    _ExtentX       =  7011
    _ExtentY       =  3201
    View          =  3
    Arrange        =  2
    LabelEdit      =  1
    MultiSelect     =  -1  'True
    LabelWrap      =  -1  'True
    HideSelection   =  -1  'True
    _Version       =  327682
    ForeColor      =  -2147483640
    BackColor      =  -2147483643
    BorderStyle     =  1
    Appearance      =  1
    NumItems       =  0
  End
  Begin VB.CheckBox chkIndexed
    Caption        =  "Indexed"
    Height        =  375
    Left         =  0
    TabIndex       =  41
    ToolTipText     =  "Check this box for indexed strings."
    Top          =  0
    Width         =  1215
  End
  Begin VB.CommandButton cmdRemove
    Caption        =  "Remove"
    Height        =  255
    Left         =  2640
    TabIndex       =  40
    ToolTipText     =  "Click here to remove a set of indexed values."
    Top          =  2520
```

```
      Width         =  1335
    End
    Begin VB.CommandButton cmdEdit
      Caption         =  "Edit"
      Height          =  255
      Left            =  1320
      TabIndex        =  39
      ToolTipText     =  "Click here to edit a set of indexed values."
      Top             =  2520
      Width           =  1335
    End
    Begin VB.CommandButton cmdAdd
      Caption         =  "Add"
      Height          =  255
      Left            =  0
      TabIndex        =  38
      ToolTipText     =  "Click here to add a new set of indexed values."
      Top             =  2520
      Width           =  1335
    End
    Begin VB.Label lblStringVals
      Caption         =  "String values"
      Height          =  255
      Left            =  0
      TabIndex        =  37
      Top             =  480
      Width           =  1695
    End
  End
  Begin VB.Frame fraUntyped
    BorderStyle     =  0 'None
    Height          =  2895
    Left            =  240
    TabIndex        =  35
    Top             =  1200
    Width           =  4815
    Begin VB.TextBox txtUntyped
      Height          =  2295
      Left            =  240
      Locked          =  -1 'True
      MultiLine       =  -1 'True
      TabIndex        =  36
      ToolTipText     =  "Interesting, no?"
      Top             =  360
      Width           =  4335
  End
End
Begin VB.Frame fraIndependent
  BorderStyle    =  0  'None
  Caption        =  "Frame1"
  Height         =  2895
  Left           =  240
  TabIndex       =  10
  Top            =  1200
  Width          =  4815
  Begin VB.CheckBox chkIsIndependent
    Caption        =  "Independent"
    Height         =  375
    Left           =  0
    TabIndex       =  11
    ToolTipText    =  "Check this box if the value of this variable is not dependent."
    Top            =  0
    Value          =  1  'Checked
    Width          =  1575
  End
  Begin VB.Frame fraRealFormat
    BorderStyle    =  0  'None
    Height         =  1095
    Left           =  0
    TabIndex       =  26
    Top            =  1680
    Width          =  4815
    Begin VB.CheckBox chkOnGrid
      Caption        =  "Value must be multiple of precision"
      Height         =  375
      Left           =  1800
      TabIndex       =  45
      Top            =  120
      Width          =  2895
    End
    Begin VB.ComboBox cboPrecision
      Height         =  315
      ItemData       =  "Variable.frx":0041
      Left           =  120
      List           =  "Variable.frx":0060
      Style          =  2  'Dropdown List
      TabIndex       =  34
      Top            =  360
      Width          =  1455
    End
```

```
      Begin VB.CheckBox chkTrailingZeros
        Caption        =  "Display trailing zeros"
        Height         =  375
        Left           =  1800
        TabIndex       =  28
        Top            =  480
        Width          =  1935
      End
      Begin VB.Label LblDecimals
        Caption        =  "Precision"
        Height         =  255
        Left           =  480
        TabIndex       =  29
        Top            =  120
        Width          =  1095
      End
    End
    Begin VB.Frame fraFractionFormat
      BorderStyle    =  0  'None
      Caption        =  "Frame1"
      Height         =  1215
      Left           =  -120
      TabIndex       =  32
      Top            =  1560
      Width          =  5055
      Begin VB.CheckBox chkMixedNumbers
        Caption        =  "Mixed numbers"
        Height         =  375
        Left           =  1560
        TabIndex       =  33
        ToolTipText    =  "Check this box if you wish improper fractions to be converted into
mixed numbers."
        Top            =  240
        Width          =  1695
      End
    End
    Begin VB.Frame fraIntRealRange
      BorderStyle    =  0  'None
      Height         =  1335
      Left           =  0
      TabIndex       =  22
      Top            =  360
      Width          =  4815
      Begin VB.TextBox txtBy
        Height         =  315
    Left          =  3240
    TabIndex       =  25
    Text          =  "1"
    ToolTipText    =  "Enter the increment here.  Variables and expressions may be used."
    Top           =  600
    Width          =  1455
  End
  Begin VB.TextBox txtTo
    Height         =  315
    Left          =  1680
    TabIndex       =  24
    Text          =  "100"
    ToolTipText    =  "Enter the value in the range here.  Variables and expressions may be
used."
    Top           =  600
    Width          =  1455
  End
  Begin VB.TextBox txtFrom
    Height         =  315
    Left          =  120
    TabIndex       =  23
    Text          =  "1"
    ToolTipText    =  "Enter the lowest value in the range here.  Variables and expressions
may be used."
    Top           =  600
    Width          =  1455
  End
  Begin VB.Label lblBy
    Caption        =  "By"
    Height         =  255
    Index          =  0
    Left          =  3840
    TabIndex       =  31
    Top           =  360
    Width          =  495
  End
  Begin VB.Label lblTo
    Caption        =  "To"
    Height         =  255
    Index          =  0
    Left          =  2280
    TabIndex       =  30
    Top           =  360
    Width          =  615
  End
```

```
      Begin VB.Label lblFrom
        Caption        =  "From"
        Height         =  255
        Index          =  0
        Left           =  720
        TabIndex       =  27
        Top            =  360
        Width          =  975
      End
    End
    Begin VB.Frame fraFractionRange
      BorderStyle     =  0 'None
      Height          =  1455
      Left            =  0
      TabIndex        =  12
      Top             =  360
      Width           =  4815
      Begin VB.TextBox txtByNum
        Height         =  315
        Left           =  3240
        TabIndex       =  18
        Text           =  "1"
        ToolTipText    =  "Enter the numerator of the increment here."
        Top            =  360
        Width          =  1455
      End
      Begin VB.TextBox txtToNum
        Height         =  315
        Left           =  1680
        TabIndex       =  17
        Text           =  "100"
        ToolTipText    =  "Enter the numerator of the highest value in the range here."
        Top            =  360
        Width          =  1455
      End
      Begin VB.TextBox txtFromNum
        Height         =  315
        Left           =  120
        TabIndex       =  16
        Text           =  "1"
        ToolTipText    =  "Enter the numerator of the lowest value of the range here."
        Top            =  360
        Width          =  1455
      End
      Begin VB.TextBox txtFromDen
```

```
  Height       =  315
  Left        =  120
  TabIndex      =  15
  Text        =  "1"
  ToolTipText    =  "Enter the denominator of the lowest value in the range here."
  Top         =  840
  Width        =  1455
End
Begin VB.TextBox txtToDen
  Height       =  315
  Left        =  1680
  TabIndex      =  14
  Text        =  "1"
  ToolTipText    =  "Enter the denominator of the highest value in the range here."
  Top         =  840
  Width        =  1455
End
Begin VB.TextBox txtByDen
  Height       =  315
  Left        =  3240
  TabIndex      =  13
  Text        =  "1"
  ToolTipText    =  "Enter the denominator of the increment here."
  Top         =  840
  Width        =  1455
End
Begin VB.Label lblBy
  Caption       =  "By"
  Height       =  255
  Index        =  1
  Left        =  3840
  TabIndex      =  21
  Top         =  120
  Width        =  255
End
Begin VB.Label lblTo
  Caption       =  "To"
  Height       =  255
  Index        =  1
  Left        =  2280
  TabIndex      =  20
  Top         =  120
  Width        =  375
End
Begin VB.Label lblFrom
```

```
      Caption      =  "From"
      Height       =  255
      Index        =  1
      Left         =  480
      TabIndex     =  19
      Top          =  120
      Width        =  495
    End
    Begin VB.Line Line1
      BorderWidth  =  3
      Index        =  0
      X1           =  120
      X2           =  1560
      Y1           =  750
      Y2           =  750
    End
    Begin VB.Line Line1
      BorderWidth  =  3
      Index        =  1
      X1           =  1680
      X2           =  3120
      Y1           =  750
      Y2           =  750
    End
    Begin VB.Line Line1
      BorderWidth  =  3
      Index        =  2
      X1           =  3240
      X2           =  4680
      Y1           =  750
      Y2           =  750
    End
  End
End
Begin VB.Label lblVarType
  Caption      =  "Type"
  Height       =  255
  Left         =  2040
  TabIndex     =  8
  Top          =  120
  Width        =  1095
End
Begin VB.Label lblVarName
  Caption      =  "Variable Name"
  Height       =  255
```

```
      Left          =  240
      TabIndex      =  3
      Top           =  120
      Width         =  1095
    End
    Begin VB.Menu mnuString
      Caption       =  "String"
      Visible       =  0  'False
      Begin VB.Menu mnuStringAdd
        Caption       =  "Add"
      End
      Begin VB.Menu mnuStringEdit
        Caption       =  "Edit"
      End
      Begin VB.Menu mnuStringRemove
        Caption       =  "Remove"
      End
    End
  End
  Attribute VB_Name = "frmVariable"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = True
  Attribute VB_Exposed = False
  Option Explicit

  Private mudtVar As Variable
  Private mudtVarInt As VarInteger
  Private mudtVarReal As VarReal
  Private mudtVarFraction As VarFraction
  Private mudtVarString As VarString
  Private mudtVarUntyped As VarUntyped

  ' to see if the variable type has changed
  Private mudtType As VariableType
  Private mudtOldType As VariableType

  ' needed for string list box
  Private mbytAddEditFlag As Byte

  ' needed for listbox update
  Private mlstListBox As ListBox

  'current active model
  Private mudtModel As Model
```

```
Public Property Let AddEditFlag(ByVal bytNewValue As Byte)

   mbytAddEditFlag = bytNewValue

End Property

Public Property Get AddEditFlag() As Byte

   AddEditFlag = mbytAddEditFlag

End Property

Public Property Let Variable(ByVal udtNewValue As Variable)

   Set mudtVar = udtNewValue

End Property

Public Property Let ListBox(ByVal lstNewValue As ListBox)

   Set mlstListBox = lstNewValue

End Property

Public Property Let Model(ByVal udtNewValue As Model)

   Set mudtModel = udtNewValue

End Property

Private Sub chkIndexed_Click()

   Call CopyListView(lvwStrings, lvwTemp)
   Call CopyListView(lvwDummy, lvwStrings)
   Call CopyListView(lvwTemp, lvwDummy)

End Sub

Private Sub CopyListView(ByVal lvw1 As ListView, lvw2 As ListView)

   Dim intI As Integer
   Dim intI2 As Integer
   Dim lsiItem As ListItem

   ' copy visible listview into temp listview
```

```
    lvw2.ListItems.Clear
    lvw2.ColumnHeaders.Clear

    For intI = 1 To lvw1.ColumnHeaders.Count
      Call lvw2.ColumnHeaders.Add(, , lvw1.ColumnHeaders(intI))
    Next intI

    For intI = 1 To lvw1.ListItems.Count
      Set lsiItem = lvw2.ListItems.Add(, , lvw1.ListItems.Item(intI).Text)
      For intI2 = 1 To lvw1.ColumnHeaders.Count - 1
        lsiItem.SubItems(intI2) = lvw1.ListItems.Item(intI).SubItems(intI2)
      Next intI2
    Next intI

End Sub

Private Sub cmdAdd_Click()

    Call mnuStringAdd_Click

End Sub

Private Sub cmdEdit_Click()

    Call mnuStringEdit_Click

End Sub

Private Sub cmdRemove_Click()

    Call mnuStringRemove_Click

End Sub

Private Sub Form_Load()

    Dim udtWAPI As New Win32API

    ' enable full row select
    Call udtWAPI.EnableListViewFullRowSelect(lvwStrings)

    ' load up explanation of untyped variables
    txtUntyped = LoadResString(1)

'   cboVarDelimiter.ListIndex = 0 ' default to "@"
```

```
cboPrecision.ListIndex = 1 ' default to ".01"

cdlCD.CancelError = True

If mbytAddEditFlag = aeEdit Then

   txtVariableName = mudtVar.name

   If mudtVar.Checksum Then
      chkChecksum = 1
   Else
      chkChecksum = 0
   End If

   Select Case TypeName(mudtVar)

      Case "VarInteger"
         Set mudtVarInt = mudtVar
         With mudtVarInt
            txtFrom = .From
            txtTo = .Too
            txtBy = .By
            If .IsIndependent Then
               chkIsIndependent = 1
            Else
               chkIsIndependent = 0
            End If
         End With
         mudtType = vtInteger

      Case "VarReal"
         Set mudtVarReal = mudtVar
         With mudtVarReal
            txtFrom = .From
            txtTo = .Too
            txtBy = .By
            If .IsIndependent Then
               chkIsIndependent = 1
            Else
               chkIsIndependent = 0
            End If
            If .IsOnGrid Then
               chkOnGrid = 1
            Else
               chkOnGrid = 0
            End If
            If .TrailingZeros Then
               chkTrailingZeros = 1
            Else
               chkTrailingZeros = 0
            End If
            cboPrecision = .Precision
         End With
         mudtType = vtReal

      Case "VarFraction"
         Set mudtVarFraction = mudtVar
         With mudtVarFraction
            txtFromNum = .FromNumerator
            txtFromDen = .FromDenominator
            txtToNum = .ToNumerator
            txtToDen = .ToDenominator
            txtByNum = .ByNumerator
            txtByDen = .ByDenominator
            If .IsIndependent Then
               chkIsIndependent = 1
            Else
               chkIsIndependent = 0
            End If
            If .MixedNumbers Then
               chkMixedNumbers = 1
            Else
               chkMixedNumbers = 0
            End If
         End With
         mudtType = vtFraction

      Case "VarString"
         Set mudtVarString = mudtVar
         With mudtVarString
         mudtType = vtString
            If .Delimiter = Chr(STRING_DELIMITER) Then
               ' do nothing
            Else
               ConvertDelimiter
               .Delimiter = Chr(STRING_DELIMITER)
            End If
            ' load list view control
            If .IsIndexed Then
               chkIndexed = 1
```

```
                    Else
                        chkIndexed = 0
                    End If
                    LoadListView
                End With

            Case "VarUntyped"
                Set mudtVarUntyped = mudtVar
                mudtType = vtUntyped

        End Select

        mudtOldType = mudtType
        cboVarType.ListIndex = mudtType 'generates a cboVarType_Click event

    Else ' it's an add

        mudtType = vtInteger
        mudtOldType = mudtType
        cboVarType.ListIndex = vtInteger 'generates a cboVarType_Click event

    End If

    ' changes control states if model is frozen
    UpdateControlStates


End Sub

Private Sub cmdVarOK_Click()

    ' will capitalize the first letter of the variable name, if it's not
    ' capitalized already.
    txtVariableName_LostFocus

    ' make sure all input is valid, otherwise, make 'em fix it!
    If ValidateForm = False Then
        Exit Sub
    End If

    If mbytAddEditFlag = aeEdit Then ' we're editing an old one
        Call ProcessEdit
    Else
        Call ProcessAdd
    End If
```

```
   Unload Me

End Sub

Private Sub cmdVarCancel_Click()

   Unload Me

End Sub

Private Sub cmdVarImport_Click()

   Dim strFN As String

   With cdlCD
      .FileName = ""
      .DialogTitle = "Import strings from file"
      .Filter = "String Files (*.str)|*.str|"
      .DefaultExt = ".str"
      .InitDir = "c:\tcs\tca\strings"
      .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
      On Error GoTo Cancel
      .ShowOpen
      On Error GoTo 0
      strFN = .FileName
   End With

   On Error GoTo BeatIt ' trap open, I/O errors

   Open strFN For Input Access Read As 1

   Dim varR As Variant
   Dim varIndexed As Variant
   Dim varNumIndices As Variant
   Dim strMessage As String
   Dim mcolStr As Collection
   Dim intI As Integer

   Input #1, varIndexed

   If varIndexed Then
      strMessage = "indexed."
   Else
      strMessage = "not indexed."
```

```
    End If

    If varIndexed <> chkIndexed Then
      Call MsgBox("Unable to import: file contains string values that are " & _
        strMessage, vbExclamation, "Error")
      GoTo BeatIt
    End If

    Input #1, varNumIndices

    Do
      Input #1, varR
      If varIndexed Then
        Set mcolStr = New Collection
        Call mcolStr.Add(varR)
        For intI = 1 To varNumIndices - 1
          Input #1, varR
          Call mcolStr.Add(varR)
        Next intI
        Call AddColToListView(mcolStr)
      Else
        Call AddStrToListView(varR)
      End If

    Loop Until EOF(1)

  BeatIt:
    Close 1

  Cancel:
    Exit Sub

  End Sub

  Private Sub cmdVarExport_Click()

    Dim strFN As String

    cdlCD.CancelError = True

    With cdlCD
      .FileName = ""
      .DialogTitle = "Export strings to file"
      .Filter = "String Files (*.str)|*.str|"
      .DefaultExt = ".txt"
```

```
        .InitDir = "c:\tcs\tca\strings"
        .Flags = cdlOFNOverwritePrompt + cdlOFNHideReadOnly
        On Error GoTo Cancel
        .ShowSave
        On Error GoTo 0
        strFN = .FileName
    End With

    On Error GoTo BeatIt

    Open strFN For Output Access Write As 1

    Dim varW As Variant

    varW = chkIndexed ' so we can tell if it's indexed
    Print #1, varW
    varW = lvwStrings.ColumnHeaders.Count ' how many indices
    Print #1, varW

    Dim intI As Integer
    Dim intI2 As Integer
    Dim lsiItem As ListItem

    intI = 1

    Do ' write the data
        Set lsiItem = lvwStrings.ListItems.Item(intI)
        varW = lsiItem.Text
        Print #1, varW

        If chkIndexed Then
            For intI2 = 1 To lvwStrings.ColumnHeaders.Count - 1
                varW = lsiItem.SubItems(intI2)
                Print #1, varW
            Next intI2
        End If

        intI = intI + 1

    Loop Until intI > lvwStrings.ListItems.Count

BeatIt:
    Close 1

Cancel:
    Exit Sub

End Sub

Private Sub lvwStrings_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

    If Button = vbRightButton Then
        PopupMenu mnuString
    End If

End Sub

Private Sub mnuStringAdd_Click()

    If chkIndexed Then
        With frmIndexedString
            ' set the model
            .Model = mudtModel
            ' set the edit flag
            .AddEditFlag = aeAdd
            ' set var name
            .VariableName = txtVariableName
            ' do it
            .Show vbModal
            If .OK Then
                Call AddColToListView(.SubStringCollection)
            End If
        End With
    Else
        With frmString
            ' set the model
            .Model = mudtModel
            ' set the string
            .StringValue = ""
            ' set var name
            .VariableName = txtVariableName
            ' do it
            .Show vbModal
            If .OK Then
                Call AddStrToListView(.StringValue)
            End If
        End With
    End If
    UpdateControlStates

End Sub

Private Sub mnuStringEdit_Click()

    Dim colC As Collection

    If lvwStrings.SelectedItem Is Nothing Then Exit Sub ' Make sure list item is selected

    If chkIndexed Then
        With frmIndexedString
            ' set the model
            .Model = mudtModel
            ' set the edit flag
            .AddEditFlag = aeEdit
            ' set the substring collection
            .SubStringCollection = GetSubStringCollection(lvwStrings.SelectedItem)
            ' set var name
            .VariableName = txtVariableName
            ' do it
            .Show vbModal
            If .OK Then
                Call UpdateListView(lvwStrings.SelectedItem, .SubStringCollection)
            End If
        End With
    Else
        With frmString
            ' set the model
            .Model = mudtModel
            ' set the string
            .StringValue = lvwStrings.SelectedItem
            ' set var name
            .VariableName = txtVariableName
            ' do it
            .Show vbModal
            If .OK Then
                Set colC = New Collection
                Call colC.Add(.StringValue)
                Call UpdateListView(lvwStrings.SelectedItem, colC)
            End If
        End With
    End If

End Sub
```

```
Private Sub mnuStringRemove_Click()

  If lvwStrings.SelectedItem Is Nothing Then Exit Sub

  If MsgBox("Remove string value " & lvwStrings.SelectedItem.Text & "?", _
    vbQuestion + vbYesNo) = vbNo Then
    Exit Sub
  End If

  With lvwStrings
    Call .ListItems.Remove(.SelectedItem.index)
  End With

  UpdateControlStates

End Sub

Private Sub chkIsIndependent_Click()

  Call FormatForm

End Sub

Private Sub cboVarType_Click()

  mudtType = cboVarType.ListIndex

  Call FormatForm

End Sub

Private Sub txtVariableName_GotFocus()

  ' Automatically select all text when TextBox gets focus
  Call txtSelectAll(txtVariableName)

End Sub

Private Sub txtVariableName_LostFocus()

  Dim strName As String
  Dim udtVar As Variable

  ' Capitalize the variable name in the textbox
  strName = txtVariableName
```

```
    Call CapitalizeString(strName)
    txtVariableName = strName

End Sub

Private Sub txtFrom_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtFrom)

End Sub

Private Sub txtTo_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtTo)

End Sub

Private Sub txtBy_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtBy)

End Sub

Private Sub txtFromNum_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtFromNum)

End Sub

Private Sub txtFromDen_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtFromDen)

End Sub

Private Sub txtToNum_GotFocus()

    ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtToNum)
```

```
End Sub

Private Sub txtToDen_GotFocus()

   ' Automatically select all text when TextBox gets focus
   Call txtSelectAll(txtToDen)

End Sub

Private Sub txtByNum_GotFocus()

   ' Automatically select all text when TextBox gets focus
   Call txtSelectAll(txtByNum)

End Sub

Private Sub txtByDen_GotFocus()

   ' Automatically select all text when TextBox gets focus
   Call txtSelectAll(txtByDen)

End Sub

Private Sub FormatForm()

   cmdVarImport.Visible = False
   cmdVarExport.Visible = False

   chkIsIndependent.TabStop = False
   txtFrom.TabStop = False
   txtTo.TabStop = False
   txtBy.TabStop = False
   txtFromNum.TabStop = False
   txtFromDen.TabStop = False
   txtToNum.TabStop = False
   txtToDen.TabStop = False
   txtByNum.TabStop = False
   txtByDen.TabStop = False
   lvwStrings.TabStop = False
   chkTrailingZeros.TabStop = False
   chkTrailingZeros.TabStop = False
   chkMixedNumbers.TabStop = False

   Select Case mudtType
```

```
Case vtInteger
   fraFractionRange.Visible = False
   fraFractionFormat.Visible = False
   fraIndependent.ZOrder
   fraIntRealRange.ZOrder
   fraRealFormat.Visible = False
   chkIsIndependent.TabStop = True
   If chkIsIndependent Then
      fraIntRealRange.Visible = True
      txtFrom.TabStop = True
      txtTo.TabStop = True
      txtBy.TabStop = True
   Else
      fraIntRealRange.Visible = False
   End If

Case vtReal
   fraFractionRange.Visible = False
   fraFractionFormat.Visible = False
   fraIndependent.ZOrder
   fraIntRealRange.ZOrder
   fraRealFormat.ZOrder
   fraRealFormat.Visible = True
   chkIsIndependent.TabStop = True
   If chkIsIndependent Then
      fraIntRealRange.Visible = True
      txtFrom.TabStop = True
      txtTo.TabStop = True
      txtBy.TabStop = True
   Else
      fraIntRealRange.Visible = False
   End If
   chkOnGrid.TabStop = True
   chkTrailingZeros.TabStop = True

Case vtFraction
   fraIntRealRange.Visible = False
   fraRealFormat.Visible = False
   fraIndependent.ZOrder
   fraFractionRange.ZOrder
   fraFractionFormat.ZOrder
   fraFractionFormat.Visible = True
   chkIsIndependent.TabStop = True
   If chkIsIndependent Then
      fraFractionRange.Visible = True
```

```
            txtFromNum.TabStop = True
            txtFromDen.TabStop = True
            txtToNum.TabStop = True
            txtToDen.TabStop = True
            txtByNum.TabStop = True
            txtByDen.TabStop = True
          Else
            fraFractionRange.Visible = False
          End If
          chkMixedNumbers.TabStop = True

        Case vtString
          fraString.ZOrder
          cmdVarImport.Visible = True
          cmdVarExport.Visible = True

        Case vtUntyped
          fraUntyped.ZOrder

      End Select

      Dim intTabIndex As Integer

      intTabIndex = 4

      Call AddTab(chkIsIndependent, intTabIndex)
      Call AddTab(txtFrom, intTabIndex)
      Call AddTab(txtTo, intTabIndex)
      Call AddTab(txtBy, intTabIndex)
      Call AddTab(txtFromNum, intTabIndex)
      Call AddTab(txtFromDen, intTabIndex)
      Call AddTab(txtToNum, intTabIndex)
      Call AddTab(txtToDen, intTabIndex)
      Call AddTab(txtByNum, intTabIndex)
      Call AddTab(txtByDen, intTabIndex)
      Call AddTab(chkTrailingZeros, intTabIndex)
      Call AddTab(chkOnGrid, intTabIndex)
      Call AddTab(chkMixedNumbers, intTabIndex)

    End Sub

    ' add a tab, if its active
    Private Sub AddTab(ByVal ctlC As Control, intIndex As Integer)

      If ctlC.TabStop Then
```

```
        ctlC.TabIndex = intIndex
        intIndex = intIndex + 1
    End If

End Sub

Private Function ValidateForm() As Boolean

    ValidateForm = False

    ' check variable name length > 0
    If Len(txtVariableName) = 0 Then
        Call MsgBox("Variable names must be 1 or more characters long.", _
            vbExclamation, "Error")
        txtVariableName.SetFocus
        Exit Function
    End If

    'check first character for alpha
    If Asc(txtVariableName) < 65 Or Asc(txtVariableName) > 91 Then
        Call MsgBox("Variable names must begin in a letter", _
            vbExclamation, "Error")
        txtVariableName.SetFocus
        Exit Function
    End If

    ' check for unique variable name
    Dim blnUnique As Boolean
    blnUnique = True

    Select Case mbytAddEditFlag

        Case aeAdd
            blnUnique = mudtModel.Variables.UniqueName(txtVariableName)

        Case aeEdit
            blnUnique = mudtModel.Variables.UniqueName(txtVariableName, 1, mudtVar)

    End Select

    If blnUnique = False Then
        Call MsgBox("Variable name is already in use.", vbExclamation, "Error")
        txtVariableName.SetFocus
        Exit Function
    End If
    ' if integer or real, validate contents of From, To, By
    If cboVarType = "Integer" Or cboVarType = "Real" Then
      If Not ValidateRange Then
        Call MsgBox("Entries in From, To, and By must be either a number " & _
          "or a string variable containing a numeric value.  " & _
          "Expressions or math variables are not permitted.", _
          vbExclamation, "Error")
        Exit Function
      End If
    End If

    ValidateForm = True

  End Function

  Private Function ValidateRange() As Boolean

    Dim conC As Control
    Dim colC As New Collection
    Dim udtV As Variable
    Dim udtVS As VarString
    Dim intI As Integer
    Dim blnOK As Boolean

    Call colC.Add(txtFrom)
    Call colC.Add(txtTo)
    Call colC.Add(txtBy)

    For Each conC In colC
      blnOK = False
      If IsNumeric(conC) Then
        blnOK = True
      Else ' see if the box contains a string variable
        For Each udtV In mudtModel.Variables
          If udtV.Typ = vtString Then
            Set udtVS = udtV
            If udtVS.IsIndexed Then
              For intI = 1 To udtVS.NumIndices
                If conC = GetIndexedName(udtV.name, intI) Then
                  blnOK = True
                  Exit For
                End If
              Next intI
            ElseIf conC = udtV.name Then
```

```
                blnOK = True
            End If
        End If
        If blnOK Then
            Exit For
        End If
    Next udtV
  End If
  If Not blnOK Then
    ValidateRange = False
    Exit Function
  End If
Next conC

ValidateRange = True

End Function
Private Sub ProcessEdit()

  ' Check to see if the type has changed
  If mudtType <> mudtOldType Then

    With mlstListBox
      ' remove the old variable from the collection
      Call mudtModel.Variables.Remove(Str(.ItemData(.ListIndex)))
      ' add the new variable
      Call AddVariable
      ' update the index in the list box
      .ItemData(.ListIndex) = mudtVar.index
      ' replace the text in the list box
      .List(.ListIndex) = mudtVar.ScreenFormat
    End With

  Else
    ' update it with new data from form
    Select Case mudtType

      Case vtInteger
        Call mudtVarInt.Update(txtVariableName, _
          txtFrom, txtTo, txtBy, _
          chkIsIndependent, chkChecksum)

      Case vtReal
        Call mudtVarReal.Update(txtVariableName, _
          txtFrom, txtTo, txtBy, chkIsIndependent, _
```

```
                chkChecksum, chkTrailingZeros.Value, cboPrecision, chkOnGrid)

            Case vtFraction
                Call mudtVarFraction.Update(txtVariableName, _
                    txtFromNum, txtFromDen, txtToNum, txtToDen, _
                    txtByNum, txtByDen, chkIsIndependent, chkChecksum, _
                    chkMixedNumbers)

            Case vtString
                Dim intI As Integer
                Dim intI2 As Integer
                Dim colStr As Collection
                Dim udtSS As SubString

                mudtVar.name = txtVariableName
                mudtVar.Checksum = chkChecksum
                mudtVarString.IsIndexed = chkIndexed

                ' build a new collection of strings
                Set colStr = New Collection
                With lvwStrings
                    For intI = 1 To (.ListItems.Count)
                        Set udtSS = New SubString
                        udtSS.Delimiter = mudtVarString.Delimiter
                        Call udtSS.AddSubString(.ListItems.Item(intI).Text)
                        For intI2 = 1 To .ColumnHeaders.Count - 1
                            Call udtSS.AddSubString(.ListItems.Item(intI).SubItems(intI2))
                        Next intI2
                        Call colStr.Add(udtSS.StringValue)
                    Next intI
                End With
                mudtVarString.StringCollection = colStr

        End Select

        With mlstListBox
            ' replace the text in the list box
            .List(.ListIndex) = mudtVar.ScreenFormat
        End With

    End If

End Sub

Private Sub ProcessAdd()
```

```
    Call AddVariable

    With mlstListBox
        ' Add the new variable to the variable list box
        Call .AddItem(mudtVar.ScreenFormat)
        ' Set ItemData to index value of the variable object
        .ItemData(.ListCount - 1) = mudtVar.index
        ' Check the check box
        .Selected(.ListCount - 1) = True
    End With

End Sub

Private Sub AddVariable()

    ' Add the new variable
    Select Case mudtType

        Case vtInteger
            Set mudtVar = mudtModel.Variables.AddInteger(txtVariableName, _
                True, txtFrom, txtTo, txtBy, chkIsIndependent, _
                chkChecksum)

        Case vtReal
            Set mudtVar = mudtModel.Variables.AddReal(txtVariableName, _
                True, txtFrom, txtTo, txtBy, chkIsIndependent, _
                chkChecksum, chkTrailingZeros.Value, cboPrecision, chkOnGrid)

        Case vtFraction
            Set mudtVar = mudtModel.Variables.AddFraction(txtVariableName, _
                True, txtFromNum, txtFromDen, txtToNum, txtToDen, _
                txtByNum, txtByDen, chkIsIndependent, chkChecksum, _
                chkMixedNumbers)

        Case vtString
            Dim intI As Integer
            Dim intI2 As Integer
            Dim colStr As New Collection
            Dim udtSS As SubString

            With lvwStrings
                For intI = 1 To (.ListItems.Count)
                    Set udtSS = New SubString
                    udtSS.Delimiter = Chr(STRING_DELIMITER)
                    udtSS.AddSubString (.ListItems.Item(intI).Text)
                For intI2 = 1 To .ColumnHeaders.Count - 1
                   Call udtSS.AddSubString(.ListItems.Item(intI).SubItems(intI2))
                Next intI2
             Call colStr.Add(udtSS.StringValue)
                Next intI
          End With
          Set mudtVar = mudtModel.Variables.AddString(txtVariableName, True, _
          chkChecksum, Chr(STRING_DELIMITER), chkIndexed, colStr)

       Case vtUntyped
          Set mudtVar = mudtModel.Variables.AddUntyped(txtVariableName, True, _
             chkChecksum)

    End Select

End Sub

Private Sub UpdateControlStates()

   Dim conC As Control

   On Error Resume Next

   ' shut off all controls that have an enabled property
   For Each conC In Me
      If mudtModel.IsFrozen Then
         conC.Enabled = False
      Else
         conC.Enabled = True
      End If
   Next conC

   On Error GoTo 0

   ' these stay on even if model is frozen
   cmdVarCancel.Enabled = True
   fraString.Enabled = True
   lvwStrings.Enabled = True
   cmdEdit.Enabled = True
   mnuStringEdit.Enabled = True

   ' if model is frozen, change caption of edit button, menu to browse
   If mudtModel.IsFrozen Then
      cmdEdit.Caption = "Browse"
      mnuStringEdit.Caption = "Browse"
    End If

    ' turn export on if there's something to export
    cmdVarExport.Enabled = CBool(lvwStrings.ListItems.Count)

    ' shut off "edit", "remove" buttons, menus if the listview is empty
    If lvwStrings.ListItems.Count = 0 Then
        mnuStringEdit.Enabled = False
        cmdEdit.Enabled = False
        mnuStringRemove.Enabled = False
        cmdRemove.Enabled = False
    End If

End Sub

' this is used to convert version 0.6 indexed strings to version 0.7 style

Private Sub ConvertDelimiter()

    Dim colStr As Collection
    Dim varS As Variant

    With mudtVarString
        Set colStr = .StringCollection
        For Each varS In colStr
            varS = ReplaceAll(varS, .Delimiter, Chr(STRING_DELIMITER))
        Next varS
    End With

End Sub

Private Sub LoadListView()

    Dim intI As Integer
    Dim varS As Variant

    With mudtVarString
        If chkIndexed Then
            ' build column headers
            For intI = 1 To .NumIndices - 1
                Call lvwStrings.ColumnHeaders.Add(, , _
                    Str(intI), lvwStrings.Width / 4)
            Next intI
        End If
        ' fill in values
```

```
            For Each varS In .StringCollection
                Call AddStrToListView(varS)
            Next varS
        End With

    End Sub

    Private Sub AddColToListView(ByVal colS As Collection)

        Dim lsiLI As ListItem

        Set lsiLI = lvwStrings.ListItems.Add(, , "")
        Call UpdateListView(lsiLI, colS)

    End Sub

    Private Sub AddStrToListView(ByVal strS As String)

        Dim udtSS As New SubString
        Dim lsiLI As ListItem
        Dim intI As Integer

        Set lsiLI = lvwStrings.ListItems.Add(, , "")
        udtSS.Delimiter = Chr(STRING_DELIMITER)
        udtSS.StringValue = strS

        Call UpdateListView(lsiLI, udtSS.StringCollection)

    End Sub

    Private Sub UpdateListView(ByVal lsiLI As ListItem, ByVal colS As Collection)

        Dim intI As Integer
        Dim intW As Integer
        Dim strColHeading As String

        If chkIndexed Then
            intW = 4
        Else
            intW = 1
        End If
        ' expand the number of columns if there aren't enough
        For intI = lvwStrings.ColumnHeaders.Count To colS.Count - 1
            If chkIndexed Then
                strColHeading = Str(intI + 1)
            Call lvwStrings.ColumnHeaders.Add(, , strColHeading, _
               lvwStrings.Width / intW)
         Else
            strColHeading = " "
            Call lvwStrings.ColumnHeaders.Add(, , strColHeading)
         End If
      Next intI

      ' plug in the values
      For intI = 1 To colS.Count
         If intI = 1 Then
            lsiLI = colS.Item(intI)
         Else
            lsiLI.SubItems(intI - 1) = colS.Item(intI)
         End If
      Next intI

      ' get rid of anything in the list view past colS.Count
      For intI = colS.Count + 1 To lvwStrings.ColumnHeaders.Count
         If intI > 1 Then
            lsiLI.SubItems(intI - 1) = ""
         Else
            lsiLI = ""
         End If
      Next intI

      Dim blnEmpty As Boolean

      ' get rid of columns with all "" from right to left
      ' stop when first column with any string > 0 length is encountered
      For intI = lvwStrings.ColumnHeaders.Count To 1 Step -1
         For Each lsiLI In lvwStrings.ListItems
            blnEmpty = True
            If intI > 1 Then
               If lsiLI.SubItems(intI - 1) <> "" Then
                  blnEmpty = False
                  Exit For
               End If
            ElseIf lsiLI <> "" Then
               blnEmpty = False
               Exit For
            End If
         Next lsiLI
         If blnEmpty Then
            Call lvwStrings.ColumnHeaders.Remove(intI)
      Else
         Exit For
      End If
   Next intI

   Dim intI2 As Integer

   ' get rid of rows with "" in all columns from the bottom up
   For intI2 = lvwStrings.ListItems.Count To 1 Step -1
      Set lsiLI = lvwStrings.ListItems.Item(intI2)
      For intI = 1 To lvwStrings.ColumnHeaders.Count
         blnEmpty = True
         If intI > 1 Then
            If lsiLI.SubItems(intI - 1) <> "" Then
               blnEmpty = False
               Exit For
            End If
         ElseIf lsiLI <> "" Then
            blnEmpty = False
            Exit For
         End If
      Next intI
      If blnEmpty Then
         Call lvwStrings.ListItems.Remove(intI2)
      End If
   Next intI2

End Sub

Private Function GetSubStringCollection(ByVal lsiLI As ListItem) As Collection

   Dim colC As New Collection
   Dim intI As Integer

   Call colC.Add(lsiLI)

   For intI = 1 To lvwStrings.ColumnHeaders.Count - 1
      Call colC.Add(lsiLI.SubItems(intI))
   Next intI

   Set GetSubStringCollection = colC

End Function
```

```
' Application.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "TCAApplication"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit


Public Sub Run()

'   Dim udtP As New Prolog
'
'   If udtP.StartProlog("hlp4lib.p4") = False Then
'       Call MsgBox("Prolog failure on startup", vbExclamation, "Error")
'   End If
'
    frmTCA.Show

End Sub
```

```
' CClones.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "CClones"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' enable i/o
Private mudtFile As File

'to hold collection
Private mcolClones As Collection

' the sequence number appended to clone filenames
Private mintSeqNum As Integer

' is dirty
Private mblnIsDirty As Boolean

Private Sub Class_Initialize()

   'creates the collection when this class is created
   Set mcolClones = New Collection

End Sub

Private Sub Class_Terminate()

   'destroys collection when this class is terminated
   Set mcolClones = Nothing

End Sub

Public Property Get Item(vntIndexKey As Variant) As Clone

  'used when referencing an element in the collection
  'vntIndexKey contains either the Index or Key to the collection,
  'this is why it is declared as a Variant
  'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
  Set Item = mcolClones(vntIndexKey)

End Property

Public Property Get Count() As Long

  'used when retrieving the number of elements in the
  'collection. Syntax: Debug.Print x.Count
  Count = mcolClones.Count

End Property

Public Property Get NextSeqNum() As Integer

  mintSeqNum = mintSeqNum + 1
  NextSeqNum = mintSeqNum

  mblnIsDirty = True

End Property

Public Property Let SeqNum(ByVal intNewValue As Integer)

  mintSeqNum = intNewValue

  mblnIsDirty = True

End Property

Public Property Get SeqNum() As Integer

  SeqNum = mintSeqNum

End Property

Public Property Get IsDirty() As Boolean

  Dim udtClone As Clone

  ' see if any collection members are dirty
  If Not mblnIsDirty Then
    For Each udtClone In mcolClones
      If udtClone.IsDirty Then
        mblnIsDirty = True
        Exit For
```

```
            End If
        Next udtClone
    End If

    IsDirty = mblnIsDirty

End Property

Private Function NextID() As Long

    ' creates a unique index to associate a clone and a listbox
    Static lngID As Long

    lngID = lngID + 1
    NextID = lngID

End Function

Public Function Add(ByVal strFN As String, _
    Optional ByVal blnAddSeqNum = False) As Clone

    Dim udtClone As New Clone

    ' add the clone sequence number to the file name if blnAddSeqNum is True.
    If blnAddSeqNum Then
        udtClone.FileName = left(strFN, Len(strFN) - 4) & _
            Trim(Str(NextSeqNum)) & ".doc"
    Else
        udtClone.FileName = ExtractFileName(strFN)
    End If

    udtClone.Index = NextID

    ' use index of the clone as the key
    Call mcolClones.Add(udtClone, Str(udtClone.Index))

    Set Add = udtClone

End Function

Public Function AddObj(ByVal udtClone As Clone) As Clone

    udtClone.Index = NextID

    ' use index of the clone as the key
    Call mcolClones.Add(udtClone, Str(udtClone.Index))

    Set AddObj = udtClone

End Function

Public Sub Remove(vntIndexKey As Variant)

    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mcolClones.Remove vntIndexKey

    mblnIsDirty = True

End Sub

Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4

    'this property allows you to enumerate
    'this collection with the For...Each syntax
    Set NewEnum = mcolClones.[_NewEnum]

End Property

Public Sub Clear()

    ' empties the collection class

    Set mcolClones = Nothing
    Set mcolClones = New Collection

    mblnIsDirty = True

End Sub

Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
    ByVal lngEndIndex As Long)

    Set mudtFile = New File

    mudtFile.FileName = strFN
    Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)
    Set mudtFile = Nothing

End Sub

Public Sub ReadObjects()

    Dim udtClone As Clone

    On Error GoTo BeatIt

    Do Until Err.Number <> 0

        Set udtClone = New Clone
        Call udtClone.ReadObjectData(mudtFile)
        udtClone.Index = NextID
        Call mcolClones.Add(udtClone, Str(udtClone.Index))

    Loop

BeatIt:

    Exit Sub

End Sub

Public Function WriteCollection(ByVal strFN As String, _
    ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

    Set mudtFile = New File

    mudtFile.FileName = strFN
    WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

    Set mudtFile = Nothing

    mblnIsDirty = False

End Function

Public Sub WriteObjects()

    Dim udtClone As Clone

    For Each udtClone In mcolClones
```

```
        Call udtClone.WriteObjectData(mudtFile)
    Next udtClone

End Sub
```

007050.54545950

```
' CConstraints.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "CConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' enable i/o
Private mudtFile As New File

'local variable to hold collection
Private mcolConstraint As Collection

' is dirty
Private mblnIsDirty As Boolean

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

   mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

   Dim udtCon As Constraint

   For Each udtCon In mcolConstraint
      If udtCon.IsDirty Then
         mblnIsDirty = True
         Exit For
      End If
   Next udtCon

   IsDirty = mblnIsDirty

End Property

Private Sub Class_Initialize()
```

```
    'creates the collection when this class is created
    Set mcolConstraint = New Collection

End Sub

Private Sub Class_Terminate()

    'destroys collection when this class is terminated
    Set mcolConstraint = Nothing

End Sub

Public Property Get Item(vntIndexKey As Variant) As Constraint

  'used when referencing an element in the collection
  'vntIndexKey contains either the Index or Key to the collection,
  'this is why it is declared as a Variant
  'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
  Set Item = mcolConstraint(vntIndexKey)

End Property

Public Property Get Count() As Long

    'used when retrieving the number of elements in the
    'collection. Syntax: Debug.Print x.Count
    Count = mcolConstraint.Count

End Property

Public Sub AddObject(udtCon As Constraint)

    ' adds constraint objects directly to the collection

    udtCon.Index = NextID
    Call mcolConstraint.Add(udtCon, Str(udtCon.Index))

    mblnIsDirty = True

End Sub

Public Function Add(ByVal strConstraint As String, ByVal blnEnabled As Boolean, _
    ByVal udtType As ConstraintType, ByVal strComment As String) As Constraint

    'create a new object
    Dim objNewMember As Constraint
    Set objNewMember = New Constraint

    'set the properties passed into the method
    With objNewMember
        .ConstraintString = strConstraint
        .Enabled = blnEnabled
        .ConstraintType = udtType
        .Comment = strComment
        .Index = NextID
        ' add the new object to the collection
        Call mcolConstraint.Add(objNewMember, Str$(.Index))
    End With

    'return the object created
    Set Add = objNewMember
    Set objNewMember = Nothing

    mblnIsDirty = True

End Function

Public Sub Remove(vntIndexKey As Variant)

    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mcolConstraint.Remove vntIndexKey

    mblnIsDirty = True

End Sub

Public Function NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"

    'this property allows you to enumerate
    'this collection with the For...Each syntax
    Set NewEnum = mcolConstraint.[_NewEnum]

End Function

Private Function NextID() As Long
```

```
    ' creates a unique index to associate a constraint and the constraint listbox(es)
    Static lngID As Long

    lngID = lngID + 1
    NextID = lngID

End Function

' returns true if strCon is already a constraint in the collection.  Used
' when importing constraints to make sure dups are not introduced.
Public Function UniqueConstraint(ByVal strCon As String) As Boolean

    Dim udtCon As Constraint

    UniqueConstraint = True

    ' Check for duplicate constraint
    For Each udtCon In mcolConstraint
        If strCon = udtCon.ConstraintString Then
            UniqueConstraint = False
            Exit For
        End If
    Next udtCon

End Function

Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
    ByVal lngEndIndex As Long)

    mudtFile.FileName = strFN
    Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)

End Sub

Public Sub ReadObjects()

    Dim udtCon As Constraint

    On Error GoTo BeatIt

    Do Until Err.Number <> 0

        Set udtCon = New Constraint
        Call udtCon.ReadObjectData(mudtFile)
        udtCon.Index = NextID
        Call mcolConstraint.Add(udtCon, Str(udtCon.Index))

    Loop

BeatIt:

    Exit Sub

End Sub

Public Function WriteCollection(ByVal strFN As String, _
    ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

    mudtFile.FileName = strFN
    WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

    mblnIsDirty = False

End Function

Public Sub WriteObjects()

    Dim udtCon As Constraint

    For Each udtCon In mcolConstraint

        Call udtCon.WriteObjectData(mudtFile)

    Next udtCon

End Sub

Public Sub Clear(ByVal udtType As VariableType)

    ' empties the collection class of all constraints of type udtType

    Dim udtCon As Constraint

    For Each udtCon In mcolConstraint

        If udtCon.ConstraintType = udtType Then
            Call mcolConstraint.Remove(Str(udtCon.Index))
        End If

    Next udtCon
```

```
End Sub

' returns true if an enabled string variable name was used
' in any enabled constraint
Public Function StringVarNamesUsed(ByVal udtCVar As CVariables) As Boolean

   ' First create a collection of all enabled constraint strings
   Dim udtCon As Constraint
   Dim colStrings As New Collection

   For Each udtCon In mcolConstraint
      If udtCon.Enabled Then
         colStrings.Add udtCon.ConstraintString
      End If
   Next udtCon

   ' create a variable collection with variable names sorted in length
   ' from longest to shortest
   Dim udtSCVar As CVariables

   Set udtSCVar = udtCVar.SortVarNamesByLength

   ' nibble variable names out of the string collection, using enabled
   ' variable names sorted in length from longest to shortest
   Dim vntS As Variant
   Dim vntT As Variant
   Dim vntStart As Variant
   Dim udtVar As Variable

   For Each vntS In colStrings
      For Each udtVar In udtSCVar
         If udtVar.Enabled Then
            vntStart = InStr(1, vntS, udtVar.Name)
            If vntStart Then
               If udtVar.Typ = vtString Then
                  StringVarNamesUsed = True
                  Exit Function
               Else
                  vntT = vntS
                  vntS = left(vntT, vntStart - 1) & _
                     right(vntT, Len(vntT) - vntStart - _
                     Len(udtVar.Name) + 1)
               End If
            End If
            End If
        Next udtVar
    Next vntS

    StringVarNamesUsed = False

End Function
```

```
' Checksum.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "Checksum"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mcolStr As Collection

Private Sub Class_Initialize()

   Set mcolStr = New Collection

End Sub

Public Sub AddValue(ByVal strNewValue As String)

   Call mcolStr.Add(strNewValue)

End Sub

Public Function ComputeCS() As Double

   Dim n As Integer
   Dim dblCS As Double
   Dim dblSum As Double
   Dim varStr As Variant
   Dim cntr As Integer
   Dim dblT As Double

   cntr = 1

'   On Error GoTo Overflow

   For Each varStr In mcolStr
      dblSum = 0
      n = Len(varStr)
      While n > 0
         dblSum = Asc(Mid(varStr, n, 1)) * n + dblSum
```

```
            n = n - 1
         Wend
         dblCS = dblSum * cntr + dblCS
         cntr = cntr + 1
      Next varStr

   'Overflow:

      ComputeCS = dblCS

      Exit Function

   End Function
```

```
' Clone.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Clone"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

' file name (without path) of this clone
Private mstrFN As String

' hold document handle
Private mdocCloneDoc As Document

' checksum of variables
Private mdblChecksum As Double

' index
Private mlngIndex As Long

' is dirty
Private mblnIsDirty As Boolean

' has been routed to TCS
Private mbytIsRouted As Byte

' program
Private mudtProgram As Program

' domain
Private mudtDomain As Domain

' the batch id
Private mstrBatchID As String

' the target template
Private udtDeliveryMode As DeliveryMode
```

```
' pure or real model
Private mudtNature As Nature

' TDer's estimate of difficulty (1-5)
Private mbytTDEstimate As Byte

' difficulty has been calculated
Private mbytIsDifficultyCalculated As Byte

' the key
Private mstrKey As String

' the item type
Private mudtItemType As ItemType

Public Enum Domain
   doArithmetic = 0
   doAlgebra = 1
   doDataAnalysis = 2
   doGeometry = 3
End Enum

Public Enum Nature
   naPure = 0
   naReal = 1
End Enum

' difficulty estimate
Private mudtDE As DifficultyEstimate

Private Sub Class_Initialize()

   mblnIsDirty = False

End Sub

Public Property Get FileName() As String

   FileName = mstrFN

End Property

Public Property Let FileName(ByVal strNewValue As String)

   If mstrFN <> strNewValue Then
```

```
        mstrFN = strNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get CloneDoc() As Document

    Set CloneDoc = mdocCloneDoc

End Property

Public Property Let CloneDoc(ByVal docNewValue As Document)

    Set mdocCloneDoc = docNewValue

End Property

Public Property Get Checksum() As Double

    Checksum = mdblChecksum

End Property

Public Property Let Checksum(ByVal dblNewValue As Double)

    If mdblChecksum <> dblNewValue Then
        mdblChecksum = dblNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get Index() As Long

    Index = mlngIndex

End Property

Public Property Let Index(ByVal lngNewValue As Long)

    If mlngIndex <> lngNewValue Then
        mlngIndex = lngNewValue
        mblnIsDirty = True
    End If
End Property

Public Property Get IsDirty() As Boolean

   IsDirty = False

   If IsDifficultyCalculated Then ' don't check DE if difficultly hasn't been calculated!
      If mblnIsDirty Or mudtDE.IsDirty Then
         IsDirty = True
      End If
   Else
      If mblnIsDirty Then
         IsDirty = True
      End If
   End If

End Property

Public Property Get IsRouted() As Byte

   IsRouted = mbytIsRouted

End Property

Public Property Let IsRouted(ByVal bytNewValue As Byte)

   If mbytIsRouted <> bytNewValue Then
      mbytIsRouted = bytNewValue
      mblnIsDirty = True
   End If

End Property

Public Property Get Program() As Program

   Program = mudtProgram

End Property

Public Property Let Program(ByVal udtNewValue As Program)

   If mudtProgram <> udtNewValue Then
      mudtProgram = udtNewValue
      mblnIsDirty = True
```

```
    End If

End Property

Public Property Get Domain() As Domain

    Domain = mudtDomain

End Property

Public Property Let Domain(ByVal udtNewValue As Domain)

    If mudtDomain <> udtNewValue Then
        mudtDomain = udtNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get IsDifficultyCalculated() As Byte

    IsDifficultyCalculated = mbytIsDifficultyCalculated

End Property

Public Property Let IsDifficultyCalculated(ByVal bytNewValue As Byte)

    If mbytIsDifficultyCalculated <> bytNewValue Then
        mbytIsDifficultyCalculated = bytNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get TDEstimate() As Byte

    TDEstimate = mbytTDEstimate

End Property

Public Property Let TDEstimate(ByVal bytNewValue As Byte)

    If mbytTDEstimate <> bytNewValue Then
        mbytTDEstimate = bytNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get BatchID() As String

    BatchID = mstrBatchID

End Property

Public Property Let BatchID(ByVal strNewValue As String)

    If mstrBatchID <> strNewValue Then
        mstrBatchID = strNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get Key() As String

    Key = mstrKey

End Property

Public Property Let Key(ByVal strNewValue As String)

    If mstrKey <> strNewValue Then
        mstrKey = strNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get ItemType() As ItemType

    ItemType = mudtItemType

End Property

Public Property Let ItemType(ByVal udtNewValue As ItemType)

    If mudtItemType <> udtNewValue Then
        mudtItemType = udtNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get DeliveryMode() As DeliveryMode

    DeliveryMode = udtDeliveryMode

End Property

Public Property Let DeliveryMode(ByVal udtNewValue As DeliveryMode)

    If udtDeliveryMode <> udtNewValue Then
        udtDeliveryMode = udtNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get Nature() As Nature

    Nature = mudtNature

End Property

Public Property Let Nature(ByVal udtNewValue As Nature)

    If mudtNature <> udtNewValue Then
        mudtNature = udtNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get DiffEst() As DifficultyEstimate

    Set DiffEst = mudtDE

End Property

Public Property Let DiffEst(ByVal udtNewValue As DifficultyEstimate)

    Set mudtDE = udtNewValue
    mblnIsDirty = True
```

```
End Property

Public Sub OpenDoc(ByVal udtWord As MSWord, ByVal strPath As String)

  Dim udtDS As New DocStatus

  If udtDS.IsOpen(mstrFN) = False Then
    Set mdocCloneDoc = _
      udtWord.WordApp.Documents.Open(FileName:=strPath & mstrFN)
  End If

  mdocCloneDoc.Activate

End Sub

Public Sub CloseDoc()

  Dim udtDS As New DocStatus

  If udtDS.IsOpen(mstrFN) Then
    Call mdocCloneDoc.Close(wdSaveChanges) ' save changes
    Set mdocCloneDoc = Nothing
  End If

End Sub

Public Sub ReadObjectData(udtFile As File)

  Dim vField As Variant

  Call udtFile.ReadField(vField) ' returns the version stamp
  Call udtFile.ReadField(vField)
  FileName = ExtractFileName(vField)

  Call udtFile.ReadField(vField)
  Key = ExtractFileName(vField)
  Call udtFile.ReadField(vField)
  ItemType = ExtractFileName(vField)
  Call udtFile.ReadField(vField)
  Program = vField
  Call udtFile.ReadField(vField)
  Domain = vField
  Call udtFile.ReadField(vField)
  BatchID = vField
  Call udtFile.ReadField(vField)
```

```
    DeliveryMode = vField
    Call udtFile.ReadField(vField)
    Nature = vField
    Call udtFile.ReadField(vField)
    TDEstimate = vField
    Call udtFile.ReadField(vField)
    IsRouted = vField
    Call udtFile.ReadField(vField)
    IsDifficultyCalculated = vField
    Set mudtDE = Nothing
    If IsDifficultyCalculated Then
       Select Case Program
          Case prGRE
             Set mudtDE = New GREDifficultyEstimate
          Case prGMAT
             Set mudtDE = New GMATDifficultyEstimate
       End Select
       Call mudtDE.ReadObjectData(udtFile)
    End If

End Sub

Public Sub WriteObjectData(udtFile As File)

    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(ExtractFileName(mstrFN))
    Call udtFile.WriteField(Key)
    Call udtFile.WriteField(ItemType)
    Call udtFile.WriteField(Program)
    Call udtFile.WriteField(Domain)
    Call udtFile.WriteField(BatchID)
    Call udtFile.WriteField(DeliveryMode)
    Call udtFile.WriteField(Nature)
    Call udtFile.WriteField(TDEstimate)
    Call udtFile.WriteField(IsRouted)
    Call udtFile.WriteField(IsDifficultyCalculated)
    If IsDifficultyCalculated Then
       Call mudtDE.WriteObjectData(udtFile)
    End If

    mblnIsDirty = False

End Sub
```

```
' CModels.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "CModels"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

'to hold collection
Private mcolModels As Collection

Private Sub Class_Initialize()

   'creates the collection when this class is created
   Set mcolModels = New Collection

End Sub

Private Sub Class_Terminate()

   'destroys collection when this class is terminated
   Set mcolModels = Nothing

End Sub

Public Property Get Item(vntIndexKey As Variant) As Model

 'used when referencing an element in the collection
 'vntIndexKey contains either the Index or Key to the collection,
 'this is why it is declared as a Variant
 'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
 Set Item = mcolModels(vntIndexKey)

End Property

Public Property Get Count() As Long

   'used when retrieving the number of elements in the
   'collection. Syntax: Debug.Print x.Count
   Count = mcolModels.Count
```

```
End Property

Public Sub AddObject(udtMod As Model)

   ' adds model objects directly to the collection.  Use the file name as the
   ' key.

   Call mcolModels.Add(udtMod, Str(udtMod.FileName))

End Sub

Public Function AddNew(ByVal strFN As String, _
   ByVal udtItemType As ItemType) As Model

   Dim udtMod As Model
   Dim udtSMC As SMCModel
   Dim udtQC As QCModel
   Dim udtDS As DSModel

   Select Case udtItemType

      Case ptStandardMC
         Set udtSMC = New SMCModel
         Set udtMod = udtSMC

      Case ptQuantComp
         Set udtQC = New QCModel
         Set udtMod = udtQC

      Case ptDataSuff
         Set udtDS = New DSModel
         Set udtMod = udtDS

   End Select

   ' file name has full path
   udtMod.FileName = strFN
   udtMod.IsFrozen = False

   ' strip path from key
   Call mcolModels.Add(udtMod, ExtractFileName(strFN))

   Set AddNew = udtMod
```

```
End Function

Public Function AddExisting(ByVal strFN As String, _
   ByVal udtItemType As ItemType) As Model

   Dim udtMod As New Model
   Dim udtSMC As SMCModel
   Dim udtQC As QCModel
   Dim udtDS As DSModel

   Select Case udtItemType

      Case ptStandardMC
         Set udtSMC = New SMCModel
         Set udtMod = udtSMC

      Case ptQuantComp
         Set udtQC = New QCModel
         Set udtMod = udtQC

      Case ptDataSuff
         Set udtDS = New DSModel
         Set udtMod = udtDS

   End Select
   ' file name has full path
   udtMod.FileName = strFN
   Call udtMod.ReadModel

   ' strip path from key
   Call mcolModels.Add(udtMod, ExtractFileName(strFN))

   Set AddExisting = udtMod

End Function

Public Sub Remove(vntIndexKey As Variant)

   'used when removing an element from the collection
   'vntIndexKey contains either the Index or Key, which is why
   'it is declared as a Variant
   'Syntax: x.Remove(xyz)
   mcolModels.Remove vntIndexKey

End Sub
```

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"

   'this property allows you to enumerate
   'this collection with the For...Each syntax
   Set NewEnum = mcolModels.[_NewEnum]

End Property

Public Sub Clear()

   ' empties the collection class

   Set mcolModels = Nothing
   Set mcolModels = New Collection

End Sub
```

```
' Constraint.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = 0   'False
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Constraint"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Member0" ,"CloningConstraint"
Attribute VB_Ext_KEY = "Member1" ,"DifficultyConstraint"
Attribute VB_Ext_KEY = "Member2" ,"MathConstraint"
Attribute VB_Ext_KEY = "Member3" ,"VariableDefinition"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private mudtType As VariableType
Private mstrConstraint As String
Private mstrComment As String
Private mlngIndex As Long
Private mblnEnabled As Boolean
Private mblnIsDirty As Boolean

' These numbers correspond to the indices of the constraint listboxes in frmTCA
Public Enum ConstraintType
   ctVariation = 0
   ctDistractor = 1
End Enum

Public Property Get ConstraintString() As String

   ConstraintString = mstrConstraint

End Property
```

```
Public Property Let ConstraintString(ByVal strNewValue As String)

  If mstrConstraint <> strNewValue Then
    mstrConstraint = strNewValue
    mblnIsDirty = True
  End If

End Property

Public Property Get Comment() As String

  Comment = mstrComment

End Property

Public Property Let Comment(ByVal strNewValue As String)

  If mstrComment <> strNewValue Then
    mstrComment = strNewValue
    mblnIsDirty = True
  End If

End Property

Public Property Get ConstraintType() As ConstraintType

  ConstraintType = mudtType

End Property

Public Property Let ConstraintType(ByVal udtNewValue As ConstraintType)

  If mudtType <> udtNewValue Then
    mudtType = udtNewValue
    mblnIsDirty = True
  End If

End Property

Public Property Get index() As Long

  index = mlngIndex

End Property
```

```
Public Property Let index(ByVal lngNewValue As Long)

   mlngIndex = lngNewValue

End Property

Public Property Get Enabled() As Boolean

   Enabled = mblnEnabled

End Property

Public Property Let Enabled(ByVal blnNewValue As Boolean)

   If mblnEnabled <> blnNewValue Then
      mblnEnabled = blnNewValue
      mblnIsDirty = True
   End If

End Property

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

   mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

   IsDirty = mblnIsDirty

End Property

Public Sub Update(ByVal strConstraint As String, ByVal udtType As ConstraintType, _
   ByVal strComment As String)

   ConstraintString = strConstraint
   ConstraintType = udtType
   Comment = strComment

End Sub

Public Sub ReadObjectData(udtFile As File)

   Dim vField As Variant
```

```
    Call udtFile.ReadField(vField) ' read version stamp
    Call udtFile.ReadField(vField)
    ConstraintType = vField

    Call udtFile.ReadField(vField)
    Enabled = vField

    Call udtFile.ReadField(vField)
    ConstraintString = vField

    Call udtFile.ReadField(vField)
    Comment = vField

End Sub

Public Sub WriteObjectData(udtFile As File)

    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(ConstraintType)
    Call udtFile.WriteField(Enabled)
    Call udtFile.WriteField(ConstraintString)
    Call udtFile.WriteField(Comment)

    mblnIsDirty = False

End Sub

' makes a copy of this object
Public Function Copy() As Constraint

    Dim udtC As New Constraint

    udtC.Enabled = Enabled
    udtC.index = index
    udtC.IsDirty = IsDirty
    udtC.ConstraintType = ConstraintType
    udtC.ConstraintString = ConstraintString
    udtC.Comment = Comment

    Set Copy = udtC

End Function
```

```
' ConstraintSolver.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "ConstraintSolver"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mcolVs As Collection
Private mcolVsSave As Collection
Private mcolCs As Collection
Private mcolCsSave As Collection
Private mcolValues As Collection
Private mbytDiffWeight As Byte
Private mdblChecksum As Double
Private mintIndex As Integer

Private WithEvents mwudtP As Prolog
Attribute mwudtP.VB_VarHelpID = -1
Private mlngRet As Long

Private mblnPrologIsRunning As Boolean

Public Enum SolveRequester
    srTest = 0
    srGenerate = 1
End Enum

Public Enum SolveReturn
    srNoSolutions = 0
    srSuccess = 1
    srPrologAborted = -1
    srPrologError = -2
End Enum

Private mudtSolveRequester As SolveRequester
Private Sub Class_Initialize()

   Set mcolVs = New Collection
   Set mcolVsSave = New Collection
   Set mcolCs = New Collection
   Set mcolCsSave = New Collection
   Set mcolValues = New Collection

End Sub

Private Sub Class_Terminate()

   ' Kill Prolog
   Set mwudtP = Nothing

End Sub

Public Property Let Prolog(ByVal udtNewValue As Prolog)

   Set mwudtP = udtNewValue

End Property

Public Property Let DiffWeight(ByVal bytNewValue As Byte)

   mbytDiffWeight = bytNewValue

End Property

Public Sub AddVariable(ByVal udtNewValue As Variable)

   If udtNewValue.Enabled Then
      Call mcolVs.Add(udtNewValue.Copy) ' uses a copy of the variable
      Call mcolVsSave.Add(udtNewValue.Copy)
   End If

End Sub

Public Sub AddConstraint(ByVal udtNewValue As Constraint)

   If udtNewValue.Enabled Then
      Call mcolCs.Add(udtNewValue.Copy) ' uses a copy of the constraint
      Call mcolCsSave.Add(udtNewValue.Copy) '
   End If
End Sub

Public Function GetNextValue(strVarName As String, _
  strValue As String) As Boolean

  Dim udtVal As Value

  If mintIndex <= mcolValues.Count Then
    Set udtVal = mcolValues.Item(mintIndex)
    strVarName = udtVal.VariableName
    strValue = udtVal.Value
    ' if the value is ^, replace with ^^ so Word doesn't choke
    If strValue = "^" Then strValue = "^^"
    mintIndex = mintIndex + 1
    GetNextValue = True
  Else
    GetNextValue = False
  End If

End Function

Public Sub ResetValueIndex()

  mintIndex = 1

End Sub

Public Property Get Checksum()

  Checksum = mdblChecksum

End Property

Public Function Solve(ByVal udtSolveRequester As SolveRequester) As SolveReturn

  Dim udtVal As Value
  Dim udtC As Constraint
  Dim udtV As Variable
  Dim udtVS As VarString
  Dim udtSS As StringSolver

  mudtSolveRequester = udtSolveRequester

  Set mcolValues = New Collection
  mintIndex = 1
```

```
CreateValueCollection

If mcolValues.Count = 0 Then
   Solve = srNoSolutions
   Exit Function
End If

' solve all string variables
For Each udtV In mcolVs
   If udtV.Typ = vtString Then
      Set udtVS = udtV
      ' if this variable has no strings, error
      If udtVS.StringCollection.Count = 0 Then
         Solve = srNoSolutions
         Exit Function
      End If
      Set udtSS = New StringSolver
      udtSS.StringVariable = udtVS
      Call LoadStringValues(udtVS, udtSS)
   End If
Next udtV

' resolve any nested values for all string variable names
ResolveNestedStrings

' resolve string variable names embedded in math variable ranges
ResolveStringsInMathVariables

' resolve string variable names embedded in constraints
ResolveConstraints

' set the difference weight (difference between variants)
mwudtP.DiffWeight = mbytDiffWeight

Dim blnMathToSolve As Boolean

' add non-string variables to prolog via the value object collection
For Each udtVal In mcolValues
   If Not udtVal.VariableType = vtString Then
      Call mwudtP.AddVariable(udtVal.PrologString)
      blnMathToSolve = True
   End If
Next udtVal
```

```
' add all constraints
For Each udtC In mcolCs
   Call mwudtP.AddConstraint(udtC.ConstraintString)
   blnMathToSolve = True
Next udtC

' call prolog if there are math constraints, error if no solution found
If blnMathToSolve Then
   ' get rid of the kill file if it exists
   DestroyKillFile
   mblnPrologIsRunning = True
   ' runs async, notifies this class when it's done via the Finished event
   mwudtP.SolveConstraintsRandomly
   If udtSolveRequester = srTest Then
      frmProlog.Caption = "Testing constraints"
      frmProlog.lblProlog.Caption = "Click Abort to terminate this test."
      frmProlog.Show vbModal
   Else
      Do
         DoEvents
      Loop While mblnPrologIsRunning
   End If
   If frmProlog.Abort Then
      ' create the kill file
      CreateKillFile
      Solve = srPrologAborted
      Exit Function
   End If
   ' not aborted
   Select Case mlngRet
      Case Is < 0
         Solve = srPrologError
         Call MsgBox("Prolog error: " & Str(mlngRet), vbExclamation, "Error")
         Exit Function
      Case 0
         Solve = srNoSolutions
         Exit Function
   End Select
End If

' load up values from Prolog
For Each udtVal In mcolValues
   If Not udtVal.VariableType = vtString Then
      udtVal.Value = mwudtP.Value(udtVal.VariableName)
   End If
```

```
    Next udtVal

    ' resolve string values that are math variable names
    ResolveMathVariablesInStrings

    Dim udtChecksum As New Checksum

    ' compute the checksum of values
    For Each udtVal In mcolValues
      If udtVal.Checksum Then
        Call udtChecksum.AddValue(udtVal.Value)
      End If
    Next udtVal

    mdblChecksum = udtChecksum.ComputeCS

    Solve = srSuccess

    ' restore the variable and constraint collections their original states,
    ' as substitutions may have contaminated them.
    Set mcolVs = New Collection
    Set mcolCs = New Collection

    For Each udtV In mcolVsSave
      Call mcolVs.Add(udtV.Copy)
    Next udtV

    For Each udtC In mcolCsSave
      Call mcolCs.Add(udtC.Copy)
    Next udtC

  End Function

  ' this event raised in Prolog class
  Private Sub mwudtP_Finished(ByVal lngRet As Long)

    mblnPrologIsRunning = False
    mlngRet = lngRet

    ' kill the form if this is a test
    If mudtSolveRequester = srTest Then
      frmProlog.Kill
    End If

  End Sub
```

```
Private Sub CreateValueCollection()

    Dim intI As Integer
    Dim udtV As Variable
    Dim udtVS As VarString
    Dim udtVal As Value

    For Each udtV In mcolVs
        If udtV.Typ = vtString Then
            Set udtVS = udtV
            If udtVS.IsIndexed Then
                For intI = udtVS.NumIndices To 1 Step -1
                    Set udtVal = New Value
                    udtVal.VariableName = GetIndexedName(udtV.name, intI)
                    udtVal.VariableType = udtV.Typ
                    udtVal.Checksum = udtV.Checksum
                    udtVal.PrologString = udtV.PrologFormat
                    Call mcolValues.Add(udtVal, udtVal.VariableName)
                Next intI
            Else
                Set udtVal = New Value
                udtVal.VariableName = udtV.name
                udtVal.VariableType = udtV.Typ
                udtVal.Checksum = udtV.Checksum
                udtVal.PrologString = udtV.PrologFormat
                Call mcolValues.Add(udtVal, udtVal.VariableName)
            End If
        Else
            Set udtVal = New Value
            udtVal.VariableName = udtV.name
            udtVal.VariableType = udtV.Typ
            udtVal.Checksum = udtV.Checksum
            udtVal.PrologString = udtV.PrologFormat
            Call mcolValues.Add(udtVal, udtVal.VariableName)
        End If
    Next udtV

End Sub

Private Sub LoadStringValues(ByVal udtV As Variable, _
    ByVal udtSS As StringSolver)

    Dim intI As Integer
    Dim varS As Variant
    Dim strVN As String
    Dim udtVal As Value
    Dim udtVS As VarString

    Set udtVS = udtV

    ' get the value or values (if indexed)
    If udtVS.IsIndexed Then
      intI = 1
      For Each varS In udtSS.RandomValueCollection
        strVN = GetIndexedName(udtV.name, intI)
        Set udtVal = mcolValues.Item(strVN)
        udtVal.Value = varS
        intI = intI + 1
      Next varS
    Else
      Set udtVal = mcolValues.Item(udtV.name)
      udtVal.Value = udtSS.RandomValueCollection(1)
    End If

End Sub

Private Sub ResolveNestedStrings()

    Dim blnContinue As Boolean
    Dim udtVal As Value

    Do
      blnContinue = False
      For Each udtVal In mcolValues
        If udtVal.VariableType = vtString Then
          If ResolveString(udtVal.VariableName) Then
            blnContinue = True
          End If
        End If
      Next udtVal
    Loop Until blnContinue = False

End Sub

Private Function ResolveString(ByVal strVN As String) As Boolean

    Dim udtVal As Value
    Dim udtVal2 As Value
    Dim strT As String
    ResolveString = False

    For Each udtVal In mcolValues
        If udtVal.VariableType = vtString Then
            Set udtVal2 = mcolValues.Item(strVN)
            strT = ReplaceAll(udtVal.Value, strVN, udtVal2.Value)
            If strT <> udtVal.Value Then
                udtVal.Value = strT
                ResolveString = True
            End If
        End If
    Next udtVal

End Function

Private Sub ResolveStringsInMathVariables()

    Dim udtVal As Value
    Dim udtVal2 As Value

    For Each udtVal In mcolValues
        If udtVal.VariableType = vtString Then
            For Each udtVal2 In mcolValues
                If Not udtVal2.VariableType = vtString Then
                    udtVal2.PrologString = ReplaceAll(udtVal2.PrologString, _
                        udtVal.VariableName, udtVal.Value)
                End If
            Next udtVal2
        End If
    Next udtVal

End Sub

Private Sub ResolveConstraints()

    Dim udtC As Constraint
    Dim udtVal As Value

    For Each udtVal In mcolValues
        If udtVal.VariableType = vtString Then
            For Each udtC In mcolCs
                udtC.ConstraintString = ReplaceAll(udtC.ConstraintString, _
                    udtVal.VariableName, udtVal.Value)
            Next udtC
        End If
    Next udtVal

End Sub

Private Sub ResolveMathVariablesInStrings()

    Dim udtVal As Value
    Dim udtVal2 As Value

    For Each udtVal In mcolValues
        If udtVal.VariableType = vtString Then
            For Each udtVal2 In mcolValues
                If Not udtVal2.VariableType = vtString Then
                    udtVal.Value = ReplaceAll(udtVal.Value, udtVal2.VariableName, _
                        udtVal2.Value)
                End If
            Next udtVal2
        End If
    Next udtVal

End Sub



' CVariables.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "CVariables"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Collection" ,"Variable"
Attribute VB_Ext_KEY = "Member0" ,"Variable"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

' enable i/o
Private mudtFile As File

'to hold collection
Private mcolVariable As Collection
```

```
' is dirty
Private mblnIsDirty As Boolean

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

   mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

   Dim udtVar As Variable

   For Each udtVar In mcolVariable
      If udtVar.IsDirty Then
         mblnIsDirty = True
         Exit For
      End If
   Next udtVar

   IsDirty = mblnIsDirty

End Property

Private Sub Class_Initialize()

   'creates the collection when this class is created
   Set mcolVariable = New Collection

   Set mudtFile = New File

End Sub

Private Sub Class_Terminate()

   'destroys collection when this class is terminated
   Set mcolVariable = Nothing

   'destroys the File object
   Set mudtFile = Nothing

End Sub

Public Property Get Item(vntIndexKey As Variant) As Variable
```

```
    'used when referencing an element in the collection
    'vntIndexKey contains either the Index or Key to the collection,
    'this is why it is declared as a Variant
    'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
    Set Item = mcolVariable(vntIndexKey)

End Property

Public Property Get Count() As Long

    'used when retrieving the number of elements in the
    'collection. Syntax: Debug.Print x.Count
    Count = mcolVariable.Count

End Property

Public Sub AddObject(udtVar As Variable)

    ' adds variable objects directly to the collection

    udtVar.Index = NextID
    Call mcolVariable.Add(udtVar, Str(udtVar.Index))

End Sub

Public Function AddInteger(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean) As Variable

    'create a new object
    Dim udtVar As Variable
    Dim udtVarInteger As New VarInteger

    Set udtVar = udtVarInteger

    'set the properties passed into the method
    With udtVar
        .Typ = vtInteger
        .Name = strName
        .Enabled = blnEnabled
        .Index = NextID
        .Checksum = blnChecksum
    End With

    With udtVarInteger
```

```
        .From = strFrom
        .Too = strTo
        .By = strBy
        .IsIndependent = blnIsIndependent
    End With

    ' add the new object to the collection
    Call mcolVariable.Add(udtVarInteger, Str(udtVar.Index))

    'return the object created
    Set AddInteger = udtVarInteger

End Function

Public Function AddReal(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
    ByVal blnTrailingZeros As Boolean, _
    ByVal strPrecision As String, ByVal blnOnGrid As Boolean) As Variable

    'create a new object
    Dim udtVar As Variable
    Dim udtVarReal As New VarReal

    Set udtVar = udtVarReal

    'set the properties passed into the method
    With udtVar
        .Typ = vtReal
        .Name = strName
        .Enabled = blnEnabled
        .Index = NextID
        .Checksum = blnChecksum
    End With

    With udtVarReal
        .From = strFrom
        .Too = strTo
        .By = strBy
        .IsIndependent = blnIsIndependent
        .TrailingZeros = blnTrailingZeros
        .Precision = strPrecision
        .IsOnGrid = blnOnGrid
    End With
    ' add the new object to the collection
    Call mcolVariable.Add(udtVarReal, Str(udtVar.Index))

    'return the object created
    Set AddReal = udtVarReal

End Function

Public Function AddFraction(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal strFromNum As String, ByVal strFromDen As String, _
    ByVal strToNum As String, ByVal strToDen As String, _
    ByVal strByNum As String, ByVal strByDen As String, _
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
    ByVal blnMixedNumbers As Boolean) As Variable

    'create a new object
    Dim udtVar As Variable
    Dim udtVarFraction As New VarFraction

    Set udtVar = udtVarFraction

    'set the properties passed into the method
    With udtVar
        .Typ = vtFraction
        .Name = strName
        .Enabled = blnEnabled
        .Index = NextID
        .Checksum = blnChecksum
    End With

    With udtVarFraction
        .FromNumerator = strFromNum
        .FromDenominator = strFromDen
        .ToNumerator = strToNum
        .ToDenominator = strToDen
        .ByNumerator = strByNum
        .ByDenominator = strByDen
        .IsIndependent = blnIsIndependent
        .MixedNumbers = blnMixedNumbers
    End With

    ' add the new object to the collection
    Call mcolVariable.Add(udtVarFraction, Str(udtVar.Index))

    'return the object created
    Set AddFraction = udtVarFraction

End Function

Public Function AddString(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal blnChecksum As Boolean, ByVal strDelimiter As String, _
    ByVal blnIsIndexed As Boolean, ByVal colString As Collection) As Variable

    'create a new object
    Dim udtVar As Variable
    Dim udtVarString As New VarString

    Set udtVar = udtVarString

    'set the properties passed into the method
    With udtVar
        .Typ = vtString
        .Name = strName
        .Enabled = blnEnabled
        .Index = NextID
        .Checksum = blnChecksum
    End With

    udtVarString.Delimiter = strDelimiter
    udtVarString.StringCollection = colString
    udtVarString.IsIndexed = blnIsIndexed

    ' add the new object to the collection
    Call mcolVariable.Add(udtVarString, Str(udtVar.Index))

    'return the object created
    Set AddString = udtVarString

End Function

Public Function AddUntyped(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal blnChecksum As Boolean)

    'create a new object
    Dim udtVar As Variable
    Dim udtVarUntyped As New VarUntyped

    Set udtVar = udtVarUntyped

    'set the properties passed into the method
    With udtVar
       .Typ = vtUntyped
       .Name = strName
       .Enabled = blnEnabled
       .Index = NextID
       .Checksum = blnChecksum
    End With

    ' add the new object to the collection
    Call mcolVariable.Add(udtVarUntyped, Str(udtVar.Index))

    'return the object created
    Set AddUntyped = udtVarUntyped

End Function

Public Sub Remove(vntIndexKey As Variant)

    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mcolVariable.Remove vntIndexKey

    mblnIsDirty = True

End Sub


Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"

    'this property allows you to enumerate
    'this collection with the For...Each syntax
    Set NewEnum = mcolVariable.[_NewEnum]

End Property

Private Function NextID() As Long

    ' creates a unique index to associate a variable and the variable listbox
    Static lngID As Long

    lngID = lngID + 1
    NextID = lngID

End Function

' returns true if strName is already a variable name in the collection.  If the
' optional parameter is used, the function will not check that variable for a dup.

Public Function UniqueName(ByVal strName As String, _
    Optional ByVal bytSkipThisVar As Byte = 0, _
    Optional ByVal udtSkipVar As Variable) As Boolean

    Dim udtVar As Variable

    UniqueName = True

    ' Check for duplicate variable name
    For Each udtVar In mcolVariable

        If UCase(strName) = UCase(udtVar.Name) Then
            If bytSkipThisVar = 1 Then
                If udtSkipVar.Index <> udtVar.Index Then
                    UniqueName = False
                    Exit For
                End If
            Else
                UniqueName = False
                Exit For
            End If
        End If

    Next udtVar

End Function

' Check enabled variables in collection for duplicate names.

Public Function DuplicateNames() As Boolean

    Dim udtVar1 As Variable
    Dim udtVar2 As Variable
    Dim intI1 As Integer
    Dim intI2 As Integer

    DuplicateNames = False
    For intI1 = 1 To mcolVariable.Count
      For intI2 = 1 To mcolVariable.Count
        If intI1 <> intI2 Then
          Set udtVar1 = mcolVariable.Item(intI1)
          Set udtVar2 = mcolVariable.Item(intI2)
          If udtVar1.Enabled And udtVar2.Enabled Then
            If udtVar1.Name = udtVar2.Name Then
              DuplicateNames = True
              Exit Function
            End If
          End If
        End If
      Next intI2
    Next intI1

End Function

Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
    ByVal lngEndIndex As Long)

    mudtFile.FileName = strFN
    Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)

End Sub

Public Sub ReadObjects()

    Dim udtVar As Variable
    Dim udtType As VariableType

    On Error GoTo BeatIt

    Do Until Err.Number <> 0

      Set udtVar = New Variable
      udtType = udtVar.ReadType(mudtFile)

      Select Case udtType
        Case vtInteger
          Set udtVar = New VarInteger
          udtVar.Typ = vtInteger
        Case vtReal
          Set udtVar = New VarReal
          udtVar.Typ = vtReal
        Case vtFraction
```

```
                Set udtVar = New VarFraction
                udtVar.Typ = vtFraction
             Case vtString
                Set udtVar = New VarString
                udtVar.Typ = vtString
             Case vtUntyped
                Set udtVar = New VarUntyped
                udtVar.Typ = vtUntyped
          End Select

          Call udtVar.ReadObjectData(mudtFile)
          udtVar.Index = NextID
          Call mcolVariable.Add(udtVar, Str(udtVar.Index))

       Loop

BeatIt:
          Exit Sub

End Sub

Public Function WriteCollection(ByVal strFN As String, _
    ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

    mudtFile.FileName = strFN
    WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

    mblnIsDirty = False

End Function

Public Sub WriteObjects()

    Dim udtVar As Variable

    For Each udtVar In mcolVariable
       Call udtVar.WriteObjectData(mudtFile)
    Next udtVar

End Sub

Public Sub Clear()

    ' empties the collection class
    Set mcolVariable = Nothing
    Set mcolVariable = New Collection

End Sub

' returns a collection of variables sorted by length of variable name,
' longest to shortest
Public Function SortVarNamesByLength() As CVariables

    Dim udtVar As Variable
    Dim intLen As Integer
    Dim intLongest As Integer
    Dim udtCVar As New CVariables

    ' Find longest variable name
    For Each udtVar In mcolVariable
        If udtVar.Enabled Then
            intLen = Len(udtVar.Name)
            If intLen > intLongest Then
              intLongest = intLen
            End If
        End If
    Next udtVar

    ' Sort variables by length of name - longest first
    Do
        For Each udtVar In mcolVariable
            If udtVar.Enabled Then
                intLen = Len(udtVar.Name)
                If intLen = intLongest Then
                  ' Put this var in sorted collection
                  udtCVar.AddObject udtVar
                End If
            End If
        Next udtVar
        intLongest = intLongest - 1
    Loop While intLongest > 0

    Set SortVarNamesByLength = udtCVar

End Function
```

```
' CVariants.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "CVariants"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit


'to hold collection
Private mcolVariants As Collection

Private Sub Class_Initialize()

   'creates the collection when this class is created
   Set mcolVariant = New Collection

End Sub


Private Sub Class_Terminate()

   'destroys collection when this class is terminated
   Set mcolVariant = Nothing

End Sub

Public Property Get Item(vntIndexKey As Variant) As Variant

 'used when referencing an element in the collection
 'vntIndexKey contains either the Index or Key to the collection,
 'this is why it is declared as a Variant
 'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
 Set Item = mcolVariant(vntIndexKey)

End Property

Public Property Get Count() As Long
```

```
   'used when retrieving the number of elements in the
   'collection. Syntax: Debug.Print x.Count
   Count = mcolVariant.Count

End Property

Public Sub AddObject(udtVar As Variant)

   ' adds variable objects directly to the collection

   udtVar.Index = NextID
   Call mcolVariant.Add(udtVar, Str(udtVar.Index))

End Sub

Public Function Add(ByVal strName As String, _
   ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String) As Variant

   'create a new object
   Dim udtVar As Variant
   Dim udtVarInteger As New VarInteger

   Set udtVar = udtVarInteger

   'set the properties passed into the method
   With udtVar
      .Name = strName
      .Index = NextID
   End With

   With udtVarInteger
      .From = strFrom
      .Too = strTo
      .By = strBy
   End With

   ' add the new object to the collection
   Call mcolVariant.Add(udtVarInteger, Str(udtVar.Index))

   'return the object created
   Set AddInteger = udtVarInteger

End Function
```

```
Public Sub Remove(vntIndexKey As Variant)

    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mcolVariant.Remove vntIndexKey

End Sub


Public Property Get NewEnum() As IUnknown

    'this property allows you to enumerate
    'this collection with the For...Each syntax
    Set NewEnum = mcolVariant.[_NewEnum]

End Property

Private Function NextID() As Long

    ' creates a unique index to associate a variable and the variable listbox
    Static lngID As Long

    lngID = lngID + 1
    NextID = lngID

End Function

Public Sub Clear()

    ' empties the collection class

    Set mcolVariant = Nothing
    Set mcolVariant = New Collection

End Sub
```

```
' DifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "DifficultyEstimate"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mblnIsDirty As Boolean

Private Sub Class_Initialize()

   mblnIsDirty = False

End Sub

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

   mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

   IsDirty = mblnIsDirty

End Property

' implemented in the subclasses of DifficultyEstimate
Public Function ComputeDifficulty() As Double

End Function

' implemented in the subclasses of DifficultyEstimate
Public Function Copy() As DifficultyEstimate

End Function

' implemented in the subclasses of DifficultyEstimate
Public Sub ReadObjectData(udtFile As File)
```

```
End Sub

' implemented in the subclasses of DifficultyEstimate
Public Sub WriteObjectData(udtFile As File)

End Sub
```

```
' DocStatus.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "DocStatus"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' returns true if this document strFN is open
Public Function IsOpen(ByVal strFN As String) As Boolean

   Dim docD As Document

   For Each docD In Documents
      If InStr(1, strFN, docD.Name) Then
         IsOpen = True
         Exit Function
      End If
   Next docD

   IsOpen = False

End Function
```

```
' DSMODEL.CLS
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "DSModel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Model

Dim mudtModel As Model

Private Sub Class_Initialize()

    Set mudtModel = New Model

End Sub

' Delegated to Class Model
Public Property Get Model_FileName() As String

    Model_FileName = mudtModel.FileName

End Property

' Delegated to Class Model
Public Property Let Model_FileName(ByVal strNewValue As String)

    mudtModel.FileName = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsFrozen() As Boolean

    Model_IsFrozen = mudtModel.IsFrozen
```

```
End Property

' Delegated to Class Model
Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)

   mudtModel.IsFrozen = blnNewValue

End Property

' Delegated to Class Model
Public Property Get Model_Comments() As String

   Model_Comments = mudtModel.Comments

End Property

' Delegated to Class Model
Public Property Let Model_Comments(ByVal strNewValue As String)

   mudtModel.Comments = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_Clones() As CClones

   Set Model_Clones = mudtModel.Clones

End Property

' Delegated to Class Model
Public Property Get Model_Variables() As CVariables

   Set Model_Variables = mudtModel.Variables

End Property

' Delegated to Class Model
Public Property Get Model_Constraints() As CConstraints

   Set Model_Constraints = mudtModel.Constraints

End Property

'Delegated to Class Model
```

```
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)

   Call mudtModel.AddChecksum(dblChecksum)

End Sub

' Delegated to Class Model
Public Sub Model_InitChecksums()

   mudtModel.InitChecksums

End Sub

' Delegated to Class Model
Public Sub Model_InitTempChecksums()

   mudtModel.InitTempChecksums

End Sub

'Delegated to Class Model
Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean

   Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)

End Function

' Delegated to Class Model
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)

   mudtModel.IsDirty = blnNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsDirty() As Boolean

   Model_IsDirty = mudtModel.IsDirty

End Property

' Delegated to Class Model
Public Property Let Model_LastClone(ByVal intNewValue As Integer)

   mudtModel.LastClone = intNewValue
```

```
End Property

' Delegated to Class Model
Public Property Get Model_LastClone() As Integer

   Model_LastClone = mudtModel.LastClone

End Property

' Delegated to Class Model
Public Sub Model_FreezeModel()

   Call mudtModel.FreezeModel

End Sub

' Delegated to Class Model
Public Sub Model_OpenDoc(ByVal udtWord As MSWord)

   Call mudtModel.OpenDoc(udtWord)

End Sub

' Delegated to Class Model
Public Sub Model_CloseDoc()

   Call mudtModel.CloseDoc

End Sub

' Delegated to Class Model
Public Sub Model_CloseAllCloneDocs()

   Call mudtModel.CloseAllCloneDocs

End Sub

' Delegated to Class Model
Public Sub Model_ReadModel()

   mudtModel.ReadModel

End Sub

' Delegated to Class Model
```

```
Public Sub Model_ReadObjects()

   mudtModel.ReadObjects

End Sub

' Delegated to Class Model
Public Sub Model_WriteModel()

   mudtModel.WriteModel

End Sub

' Delegated to Class Model
Public Sub Model_WriteObjects()

   mudtModel.WriteObjects

End Sub

' Delegated to Class Model
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
   ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
   blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

   Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _
      blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)

End Function

' implemented here
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte)

   Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _
      bytDifference, 285)

End Sub

' Delegated to Class Model
Public Sub Model_SubstituteValues(ByVal objO As Object, _
   ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
   ByVal intStartPos As Integer)
```

```
End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

  Dim rnumber As Integer
  Dim statementRange As Range
  Dim firstNSE As String
  Dim secondNSE As String

  With udtClone.CloneDoc

    rnumber = .Tables(1).Rows.Count * Rnd + 0.5
    .Tables(1).Cell(Row:=rnumber, Column:=1).Range.Copy
    firstNSE = .Tables(1).Cell(Row:=rnumber, Column:=2).Range.Text
    firstNSE = left(firstNSE, 1)

    Set statementRange = .Bookmarks("tca_fStatement").Range
    statementRange.Paste
    .Tables(1).ConvertToText
    .Bookmarks.Add name:="tca_fStatement", Range:=statementRange
    statementRange.Borders.OutsideLineStyle = wdLineStyleSingle

    ' trim hard returns at end of statement
    Dim i, n As Integer
    Dim retchr As String
    retchr = Chr$(13)

    With statementRange
      n = 0
      i = .Words.Count

      While .Words(i).Text = retchr And i > 1
        i = i - 1
        If .Words(i).Text = retchr Then
          n = n + 1
        End If
      Wend

      If n > 0 Then
        .Words(.Words.Count - n + 1).Delete Count:=n
      End If
    End With

    rnumber = .Tables(2).Rows.Count * Rnd + 0.5
    .Tables(2).Cell(Row:=rnumber, Column:=1).Range.Copy
```

```
secondNSE = .Tables(2).Cell(Row:=rnumber, Column:=2).Range.Text
secondNSE = left(secondNSE, 1)

Set statementRange = .Bookmarks("tca_sStatement").Range
statementRange.Paste
.Tables(1).ConvertToText
.Bookmarks.Add name:="tca_sStatement", Range:=statementRange
statementRange.Borders.OutsideLineStyle = wdLineStyleSingle

' trim hard returns at end of statement
With statementRange
   n = 0
   i = .Words.Count

   While .Words(i).Text = retchr And i > 1
     i = i - 1
     If .Words(i).Text = retchr Then
       n = n + 1
     End If
   Wend

   If n > 0 Then
      .Words(.Words.Count - n + 1).Delete Count:=n
   End If
End With

Dim key As String
Dim keyChr As String

If firstNSE = "N" And secondNSE = "N" Then
  key = "E"
ElseIf firstNSE = "S" And secondNSE = "S" Then
  key = "C or E"
ElseIf firstNSE = "E" And secondNSE = "E" Then
  key = "D"
ElseIf firstNSE = "N" And secondNSE = "S" Then
  key = "E"
ElseIf firstNSE = "E" And secondNSE = "S" Then
  key = "A"
ElseIf firstNSE = "S" And secondNSE = "E" Then
  key = "B"
ElseIf firstNSE = "N" And secondNSE = "E" Then
  key = "B"
ElseIf firstNSE = "E" And secondNSE = "N" Then
  key = "A"
```

```
        End If

        keyChr = left(.Bookmarks("key").Range.Text, 1)

        If keyChr = "A" Or keyChr = "1" Then
          key = "A"
        ElseIf keyChr = "B" Or keyChr = "2" Then
          key = "B"
        ElseIf keyChr = "C" Or keyChr = "3" Then
          key = "C"
        ElseIf keyChr = "D" Or keyChr = "4" Then
          key = "D"
        ElseIf keyChr = "E" Or keyChr = "5" Then
          key = "E"
        End If

        Dim keyRange As Range
        Set keyRange = .Bookmarks("tca_Key").Range

        If key = "" Then
           keyRange.InsertBefore Text:="TCA cannot determine the key"
        Else
           keyRange.InsertBefore Text:="Key is " & key
        End If

        udtClone.key = key

    End With

End Sub
```

```
' Family.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "Family"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

' enable i/o
Private mudtFile As File

' the .mdf file name of this family
Private mstrFamilyFN As String

' the program that owns this family
Private mudtProgram As Program

' the item type
Private mudtItemType As ItemType

' close/medium far classification
Private mudtProximity As Proximity

' generic/non-generic classification
Private mblnGeneric As Boolean

' accession number, if this family is based on a locked item
Private mstrAccNum As String

' the active model
Private mudtActiveModel As Model

' collection of Models
Private mudtCModels As CModels

' the collection of accepted clones
Private mudtCClones As CClones
```

```
' is dirty?
Private mblnIsDirty As Boolean

Public Enum Program
   prGRE = 0
   prGMAT = 1
   prSAT = 2
   prMR = 3
End Enum

Public Enum ItemType
   ptStandardMC = 0
   ptQuantComp = 1
   ptDataSuff = 2
End Enum

Public Enum Proximity
   prNear = 0
   prMedium = 1
   prFar = 2
End Enum

Private Enum FamilyRecordLayout
   frLocalDataIndex = 1 ' long (takes 4 bytes)
   frCloneIndex = 5 ' long
   frLocalData = 51
   frClones = 201 ' variable length
End Enum

Private Sub Class_Initialize()

   Set mudtCModels = New CModels
   Set mudtCClones = New CClones
   mblnIsDirty = False

End Sub

Public Property Get FileName() As String

   FileName = mstrFamilyFN

End Property

Public Property Let FileName(ByVal strNewValue As String)
```

```
    mstrFamilyFN = left(strNewValue, Len(strNewValue) - 4) & ".mdf"
End Property
Public Property Get Program() As Program
    Program = mudtProgram
End Property
Public Property Let Program(ByVal udtNewValue As Program)
    mudtProgram = udtNewValue
End Property
Public Property Get ItemType() As ItemType
    ItemType = mudtItemType
End Property
Public Property Let ItemType(ByVal udtNewValue As ItemType)
    mudtItemType = udtNewValue
End Property
Public Property Get Proximity() As Proximity
    Proximity = mudtProximity
End Property
Public Property Let Proximity(ByVal udtNewValue As Proximity)
    mudtProximity = udtNewValue
End Property
Public Property Get Generic() As Boolean
    Generic = mblnGeneric
End Property
```

```
Public Property Let Generic(ByVal blnNewValue As Boolean)
   mblnGeneric = blnNewValue
End Property
Public Property Get AccNum() As String
   AccNum = mstrAccNum
End Property
Public Property Let AccNum(ByVal strNewValue As String)
   mstrAccNum = strNewValue
End Property
Public Property Get ActiveModel() As Model
   Set ActiveModel = mudtActiveModel
End Property
Public Property Let ActiveModel(ByVal udtModel As Model)
   Set mudtActiveModel = udtModel
End Property
Public Property Get Models() As CModels
   Set Models = mudtCModels
End Property
Public Property Get Clones() As CClones
   Set Clones = mudtCClones
End Property
Public Property Let IsDirty(ByVal blnNewValue As Boolean)
   mblnIsDirty = blnNewValue
```

```
End Property

Private Property Get IsDirty() As Boolean

   If mudtCClones.IsDirty Or mblnIsDirty Then
      IsDirty = True
   Else
      IsDirty = False
   End If

End Property

Public Sub CloseAllCloneDocs()

   Dim udtClone As Clone

   For Each udtClone In mudtCClones
      udtClone.CloseDoc
   Next udtClone

End Sub

Public Sub ReadFamily()

   Dim udtWAPI As New Win32API

   If udtWAPI.FileExists(mstrFamilyFN) Then
      Set mudtFile = New File
      mudtFile.FileName = mstrFamilyFN
      Call mudtFile.ReadFile(Me, frLocalDataIndex, frCloneIndex)
      Set mudtFile = Nothing
      Call mudtCClones.ReadCollection(mstrFamilyFN, frCloneIndex, READ_UNTIL_EOF)
   End If

End Sub

Public Sub ReadObjects()

   Dim vField As Variant

   Call mudtFile.ReadField(vField) ' returns the version stamp
   Call mudtFile.ReadField(vField)
   Program = vField
   Call mudtFile.ReadField(vField)
```

```
    ItemType = vField
    Call mudtFile.ReadField(vField)
    Generic = vField
    Call mudtFile.ReadField(vField)
    Proximity = vField
    Call mudtFile.ReadField(vField)
    AccNum = vField

End Sub

Public Sub WriteFamily()

    Dim udtPB As New Progress

    If IsDirty Then
        Set mudtFile = New File
        mudtFile.FileName = mstrFamilyFN
        Call udtPB.Init(2, "Saving family...")
        Call mudtFile.WriteFile(Me, True, frLocalDataIndex, frLocalData)
        udtPB.Advance
        Set mudtFile = Nothing
        Call mudtCClones.WriteCollection(mstrFamilyFN, frCloneIndex, frClones)
        udtPB.Advance
    End If

    IsDirty = False

End Sub

Public Sub WriteObjects()

    Call mudtFile.WriteField(mintVERSIONSTAMP)
    Call mudtFile.WriteField(Program)
    Call mudtFile.WriteField(ItemType)
    Call mudtFile.WriteField(Generic)
    Call mudtFile.WriteField(Proximity)
    Call mudtFile.WriteField(AccNum)

End Sub
```

```
' File.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = 0   'False
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "File"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' Path and name of the file to open
Private m_sFileName As String

' File number opened
Private m_iFileNumber As Integer

' passed in by ReadFile
Private mlngEndPos As Long

' Error constants
Enum FileError
   fileOpenError = vbObjectError + 512 + 2
   fileEOFError = vbObjectError + 512 + 3
   fileReadError = vbObjectError + 512 + 4
   fileWriteError = vbObjectError + 512 + 5
   fileStopReadingError = vbObjectError + 512 + 6
End Enum

Property Get FileName() As String
Attribute FileName.VB_Description = "Name of the file to contain the task information."

   FileName = m_sFileName

End Property

Property Let FileName(ByVal sFileName As String)

   ' Should validate valid path here
```

```
    m_sFileName = sFileName

End Property

' Reads all objects from a file into the defined object
' Parameters:
Public Sub ReadFile(obj As Object, Optional ByVal lngStartIndex As Long = 0, _
    Optional ByVal lngEndIndex As Long = 0)

    Dim lngStartPos As Long

    ' Enable error handling
    On Error Resume Next

    ' Get the file number
    m_iFileNumber = FreeFile

    ' Open the file and trap any errors
    Open m_sFileName For Binary Access Read As #m_iFileNumber

    Select Case err.Number

        Case 0  ' No error
            If lngEndIndex > 0 Then
                Seek m_iFileNumber, lngEndIndex
                Get #m_iFileNumber, , mlngEndPos
            Else
                mlngEndPos = 0
            End If
            If lngStartIndex > 0 Then
                Seek m_iFileNumber, lngStartIndex
                Get #m_iFileNumber, , lngStartPos
                Seek m_iFileNumber, lngStartPos
            End If
            obj.ReadObjects ' Get the data

        Case 53    ' File not found
            ' Do nothing

        Case Else
            ' Turn off error handling here
            On Error GoTo 0

            ' Pass the error out
            err.Raise fileOpenError, "CFile::ReadFile", "Error opening file."
    End Select

    ' Close the file
    Close #m_iFileNumber

End Sub

' Reads a field from the file
' Parameters:
'  vField     field read from the file

Public Sub ReadField(vField As Variant)
    ' Set the error handler
    On Error GoTo ERR_HANDLER

    Get #m_iFileNumber, , vField

    If EOF(m_iFileNumber) Then
        ' Reached end of file
        err.Raise fileEOFError
    End If

    If mlngEndPos > 0 Then
        If mlngEndPos < Seek(m_iFileNumber) Then
            err.Raise fileStopReadingError
        End If
    End If

Exit Sub

ERR_HANDLER:
    ' Pass the error out
    Select Case err.Number

        Case fileEOFError
            Call err.Raise(err.Number, "File::ReadField", "EOF")
        Case fileStopReadingError
            Call err.Raise(err.Number, "File::ReadField", "Stop!")
        Case Else
            Call err.Raise(fileReadError, "File::ReadField", err.Description)

   End Select
End Sub

' Writes all objects to the file.
' Parameters:
'  obj       Object
Public Function WriteFile(obj As Object, _
   Optional ByVal blnKillOldFile As Boolean = False, _
   Optional ByVal lngIndexPos As Long = 0, _
   Optional ByVal lngSeekPos As Long = 1) As Long

   ' Enable error handling
   On Error Resume Next

   If blnKillOldFile Then ' assume new file, otherwise append
      Kill m_sFileName ' Kill the existing file
      err.Clear
   End If

   ' Get the file number
   m_iFileNumber = FreeFile

   ' Open the file and trap any errors
   Open m_sFileName For Binary As #m_iFileNumber

    ' write the starting file position, if lngIndexPos > 0
   If lngIndexPos > 0 Then
      Seek m_iFileNumber, lngIndexPos
      Put #m_iFileNumber, , lngSeekPos
   End If

   ' seek to starting position
   Seek m_iFileNumber, lngSeekPos

   Select Case err.Number
      Case 0  ' No error
         ' Write the data
         obj.WriteObjects

      Case Else
         ' Turn off error handling here
         On Error GoTo 0

         ' Pass the error out
         err.Raise fileOpenError, "CFile::WriteFile", _
            "Error opening file: " & err.Description
    End Select

    ' return current position
    WriteFile = Seek(m_iFileNumber)

    ' Close the file
    Close #m_iFileNumber

End Function

' Write a field to the file
' Parameters:
'   vField      field to write to the file
Public Sub WriteField(ByVal vField As Variant)
' Set the error handler
On Error GoTo ERR_HANDLER

    Put #m_iFileNumber, , vField

Exit Sub
ERR_HANDLER:
    err.Raise fileWriteError, "CFile::WriteField", _
        "Write Error: " & err.Descpription
End Sub
```

```
' FileFind.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "FileFind"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' used for finding files that fit a mask


Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
End Type

Private Const MAX_PATH = 260

Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * MAX_PATH
        cAlternate As String * 14
End Type

Private Const INVALID_HANDLE_VALUE = -1

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
    (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" _
    (ByVal hFileName As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long
Private Declare Function GetCurrentDirectory Lib "kernel32" _
   Alias "GetCurrentDirectoryA" (ByVal nBufferLength As Long, _
   ByVal lpBuffer As String) As Long

' returns true if strFN exists
Public Function Exists(ByVal strFN) As Boolean

   Dim lngHandle As Long
   Dim w32FindData As WIN32_FIND_DATA

   lngHandle = FindFirstFile(strFN, w32FindData)

   If lngHandle = INVALID_HANDLE_VALUE Then
      Exists = False
   Else
      Exists = True
      Call FindClose(lngHandle)
   End If

End Function

' returns a collection of file names that satisfy strMask.  The path seems to
' disappear from the returned file names.

Public Function FindAll(ByVal strMask As String) As Collection

   Dim lngHandle As Long
   Dim lngRet As Long
   Dim w32FindData As WIN32_FIND_DATA
   Dim strFN As String
   Dim varI As Variant
   Dim colFNs As New Collection

   lngHandle = FindFirstFile(strMask, w32FindData)

   If lngHandle = INVALID_HANDLE_VALUE Then
      Exit Function
   End If

   Do
      varI = InStr(1, w32FindData.cFileName, Chr(0)) ' trim off the nulls
      strFN = left(w32FindData.cFileName, varI - 1)
      Call colFNs.Add(strFN) ' add to the collection

   Loop Until FindNextFile(lngHandle, w32FindData) = 0
```

```
    Set FindAll = colFNs

End Function

' returns the current directory
Public Function CurrentDirectory() As String

    Dim strBuf As String
    Dim lngRet As Long
    Dim varI As Variant

    strBuf = Space(300)
    lngRet = GetCurrentDirectory(300, strBuf)
    varI = InStr(1, strBuf, Chr(0)) ' trim off the nulls
    CurrentDirectory = left(strBuf, varI - 1)

End Function
```

```
' GMATDifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "GMATDifficultyEstimate"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Implements DifficultyEstimate

Private mudtDE As DifficultyEstimate

' these go into the GMAT model
Private mudtDomain As Domain
Private mstrKey As String
Private mudtNature As Nature
Private mudtItemType As ItemType
Private mintTDDiffEst As Integer

Private Sub Class_Initialize()

    Set mudtDE = New DifficultyEstimate

End Sub

Private Sub Class_Terminate()

    Set mudtDE = Nothing

End Sub

Public Property Get DifficultyEstimate_IsDirty() As Boolean

    DifficultyEstimate_IsDirty = mudtDE.IsDirty

End Property
```

```
Public Property Let DifficultyEstimate_IsDirty(ByVal blnNewValue As Boolean)
   mudtDE.IsDirty = blnNewValue
End Property
Public Property Let Domain(ByVal udtNewValue As Domain)
   mudtDomain = udtNewValue
End Property
Public Property Let Nature(ByVal udtNewValue As Nature)
   mudtNature = udtNewValue
End Property
Public Property Let Key(ByVal strNewValue As String)
   mstrKey = strNewValue
End Property
Public Property Let ItemType(ByVal udtNewValue As ItemType)
   mudtItemType = udtNewValue
End Property
Public Property Let TDDiffEst(ByVal intNewValue As Integer)
   mintTDDiffEst = intNewValue
End Property
Public Function DifficultyEstimate_ComputeDifficulty() As Double
   Dim dblDiff As Double
   dblDiff = -2.3289902

   ' add coeff for domain
   If mudtDomain = doAlgebra Then
      dblDiff = dblDiff + 0.2341578
```

```
    ElseIf mudtDomain = doGeometry Then
      dblDiff = dblDiff + 0.3749013
    End If

    ' add coeff for real
    If mudtNature = naReal Then
      dblDiff = dblDiff + 0.3285613
    End If

    ' add coeff for td difficulty estimate
    dblDiff = dblDiff + ((6 - mintTDDiffEst) * 0.7024191)

    ' add coeff for key
    If mudtItemType = ptDataSuff Then
      If mstrKey = "A" Or mstrKey = "B" Then
        dblDiff = dblDiff + 0.7334054
      End If
    End If

    DifficultyEstimate_ComputeDifficulty = dblDiff

  End Function

  ' returns a copy of this object
  Public Function DifficultyEstimate_Copy() As DifficultyEstimate

    Dim udtGmatDE As New GMATDifficultyEstimate

    Set DifficultyEstimate_Copy = udtGmatDE

  End Function

  Public Sub DifficultyEstimate_ReadObjectData(udtFile As File)

    Dim vField As Variant

    Call udtFile.ReadField(vField) ' reads the version stamp

  End Sub

  Public Sub DifficultyEstimate_WriteObjectData(udtFile As File)

    Call udtFile.WriteField(mintVERSIONSTAMP)

    mudtDE.IsDirty = False
```

```
End Sub
```

```
' GREDifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "GREDifficultyEstimate"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Implements DifficultyEstimate

Private mudtDE As DifficultyEstimate

' these go into the GRE model
Private mudtDomain As Domain
Private mudtComputation As GREComputation
Private mudtCognition As GRECognition
Private mudtConcept As GREConcept
Private mstrKey As String
Private mudtNature As Nature
Private mudtItemType As ItemType

Public Enum GREComputation
   grIntegers = 0
   grDecimalsFractions = 1
   grRadicals = 2
   grNone = 3
End Enum

Public Enum GRECognition
   grProcedural = 0
   grConceptual = 1
   grHigherOrderThinking = 2
End Enum

Public Enum GREConcept
   grProbability = 0
   grPercentofPercent = 1
```

```
    grPercentChange = 2
    grLinearInequality = 3
    grNoneOfThese = 4
End Enum

Private Sub Class_Initialize()

    Set mudtDE = New DifficultyEstimate

End Sub

Private Sub Class_Terminate()

    Set mudtDE = Nothing

End Sub

Public Property Get DifficultyEstimate_IsDirty() As Boolean

    DifficultyEstimate_IsDirty = mudtDE.IsDirty

End Property

Public Property Let DifficultyEstimate_IsDirty(ByVal blnNewValue As Boolean)

    mudtDE.IsDirty = blnNewValue

End Property

Public Property Let Domain(ByVal udtNewValue As Domain)

    mudtDomain = udtNewValue

End Property

Public Property Get Computation() As GREComputation

    Computation = mudtComputation

End Property

Public Property Let Computation(ByVal udtNewValue As GREComputation)

    If mudtComputation <> udtNewValue Then
        mudtComputation = udtNewValue
    mudtDE.IsDirty = True
  End If

End Property

Public Property Get Cognition() As GRECognition

  Cognition = mudtCognition

End Property

Public Property Let Cognition(ByVal udtNewValue As GRECognition)

  If mudtCognition <> udtNewValue Then
    mudtCognition = udtNewValue
    mudtDE.IsDirty = True
  End If

End Property

Public Property Get Concept() As GREConcept

  Concept = mudtConcept

End Property

Public Property Let Concept(ByVal udtNewValue As GREConcept)

  If mudtConcept <> udtNewValue Then
    mudtConcept = udtNewValue
    mudtDE.IsDirty = True
  End If

End Property

Public Property Get Nature() As Nature

  Nature = mudtNature

End Property

Public Property Let Nature(ByVal udtNewValue As Nature)

  mudtNature = udtNewValue
```

```
End Property

Public Property Get Key() As String

   Key = mstrKey

End Property

Public Property Let Key(ByVal strNewValue As String)

   If mstrKey <> strNewValue Then
      mstrKey = strNewValue
      mudtDE.IsDirty = True
   End If

End Property

Public Property Get ItemType() As ItemType

   ItemType = mudtItemType

End Property

Public Property Let ItemType(ByVal udtNewValue As ItemType)

   mudtItemType = udtNewValue

End Property

Public Function DifficultyEstimate_ComputeDifficulty() As Double

   Dim dblDiff As Double

   dblDiff = 0.3296816

   ' add coeff for domain
   If mudtDomain = doAlgebra Then
      dblDiff = dblDiff + 0.2464302
   ElseIf mudtDomain = doDataAnalysis Then
      dblDiff = dblDiff - 0.3944198
   End If

   ' add coeff for computation
   If mudtComputation = grIntegers Then
      dblDiff = dblDiff - 0.8563799
```

```
    ElseIf mudtComputation = grDecimalsFractions Then
        dblDiff = dblDiff - 0.5181709
    End If

    ' add coeff for cognition
    If mudtCognition = grProcedural Then
        dblDiff = dblDiff - 0.6621277
        If mudtNature = naReal Then ' add coeff for procedural and real
            dblDiff = dblDiff - 0.8781659
        End If
    ElseIf mudtCognition = grHigherOrderThinking Then
        dblDiff = dblDiff + 0.7253093
    End If

    ' add coeff for concept
    Select Case mudtConcept
        Case grLinearInequality
            dblDiff = dblDiff - 0.5881492
        Case grNoneOfThese
            ' do nothing
        Case Else
            dblDiff = dblDiff + 0.5835095
    End Select

    ' add coeff for key
    If mudtItemType = ptQuantComp Then
        If mstrKey = "A" Or mstrKey = "B" Or mstrKey = "C" Then
            dblDiff = dblDiff - 0.531099
        End If
    End If

    DifficultyEstimate_ComputeDifficulty = dblDiff

End Function

' returns a copy of this object
Public Function DifficultyEstimate_Copy() As DifficultyEstimate

    Dim udtGreDE As New GREDifficultyEstimate

    udtGreDE.Computation = Computation
    udtGreDE.Cognition = Cognition
    udtGreDE.Concept = Concept

    Set DifficultyEstimate_Copy = udtGreDE
```

```
End Function

Public Sub DifficultyEstimate_ReadObjectData(udtFile As File)

  Dim vField As Variant

  Call udtFile.ReadField(vField) ' reads the version stamp

  Call udtFile.ReadField(vField)
  Computation = vField

  Call udtFile.ReadField(vField)
  Cognition = vField

  Call udtFile.ReadField(vField)
  Concept = vField

End Sub

Public Sub DifficultyEstimate_WriteObjectData(udtFile As File)

  Call udtFile.WriteField(mintVERSIONSTAMP)
  Call udtFile.WriteField(Computation)
  Call udtFile.WriteField(Cognition)
  Call udtFile.WriteField(Concept)

  mudtDE.IsDirty = False

End Sub



' IniFile.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "IniFile"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
' this class handles all ini file reads and writes via kernel32
```

```
Option Explicit

' the following declares are needed to get and put to .ini files
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias _
   "GetPrivateProfileSectionA" (ByVal lpAppName As String, _
   ByVal lpReturnedString As String, ByVal nSize As Long, _
   ByVal lpFileName As String) As Long

Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _
   "GetPrivateProfileStringA" (ByVal lpApplicationName As String, _
   ByVal lpKeyName As Any, ByVal lpDefault As String, _
   ByVal lpReturnedString As String, ByVal nSize As Long, _
   ByVal lpFileName As String) As Long

Private Declare Function WritePrivateProfileSection Lib "kernel32" Alias _
   "WritePrivateProfileSectionA" (ByVal lpAppName As String, _
   ByVal lpString As String, ByVal lpFileName As String) As Long

Private Declare Function WritePrivateProfileString Lib "kernel32" Alias _
   "WritePrivateProfileStringA" (ByVal lpApplicationName As String, _
   ByVal lpKeyName As Any, ByVal lpString As Any, ByVal lpFileName As String) _
   As Long

' contains file name of ini
Private mstrFN As String

' holds collection of keys created by Get ProfileSection method
Private mcolKeys As Collection

' holds collection of values created by Get ProfileSection method
Private mcolValues As Collection

Private Sub Class_Initialize()

   Set mcolKeys = New Collection
   Set mcolValues = New Collection

End Sub

' sets the ini path + file name
Public Property Let FN(ByVal strFN As String)

   mstrFN = strFN

End Property
```

```
' returns the ini path + file name
Public Property Get FN() As String

  FN = mstrFN

End Property

'gets all of the keys and values in a section
Public Sub GetProfileSection(ByVal strSectionName As String)

  Dim strRet As String
  strRet = Space(5000)

  If GetPrivateProfileSection(strSectionName, strRet, 5000, mstrFN) = 0 Then
    Call MsgBox("Ini file call unsuccessful", vbExclamation, "Error")
  End If

  Dim lngStart As Long
  Dim lngEnd As Long
  Dim str1 As String
  Dim str2 As String
  Dim varT As Variant
  Dim strT As String

  ' parse the key and variable names out of strRet, add to the collections
  For lngStart = 1 To Len(strRet)
    str1 = Mid(strRet, lngStart, 1)
    If str1 <> Chr(0) Then
      For lngEnd = lngStart + 1 To Len(strRet)
        str2 = Mid(strRet, lngEnd, 1)
        Select Case str2
          Case "="
            strT = Mid(strRet, lngStart, lngEnd - lngStart)
            Call mcolKeys.Add(strT)
            Exit For
          Case Chr(0)
            strT = Mid(strRet, lngStart, lngEnd - lngStart)
            Call mcolValues.Add(strT)
          Exit For
        End Select
      Next lngEnd
      lngStart = lngEnd
    End If
  Next lngStart
```

```
End Sub

' called after LoadProfileSection.
' sets strKey and strValue to the the KeyValue pairs if one exists
' at this index.
' returns TRUE if the index exists, FALSE if it doesn't.

Public Function GetKeyValuePair(strKey As String, strValue As String, _
   ByVal intIndex As Integer) As Boolean

   If intIndex <= mcolKeys.Count Then
      strKey = mcolKeys.Item(intIndex)
      strValue = mcolValues.Item(intIndex)
      GetKeyValuePair = True
   Else
      strKey = ""
      strValue = ""
      GetKeyValuePair = False
   End If

End Function

' init before loading key/value pairs
Public Function InitializeKeyValuePairs()

   Set mcolKeys = Nothing
   Set mcolValues = Nothing
   Set mcolKeys = New Collection
   Set mcolValues = New Collection

End Function

Public Sub SetKeyValuePair(ByVal strKey As String, ByVal strValue As String)

   Call mcolKeys.Add(strKey)
   Call mcolValues.Add(strValue)

End Sub

Public Sub WriteProfileSection(ByVal strSectionName As String)

   Dim strSection As String
   Dim varKey As Variant
   Dim varValue As Variant
   Dim intI As Integer
```

```
    For Each varKey In mcolKeys
        intI = intI + 1
        varValue = mcolValues.Item(intI)
        strSection = strSection & varKey & "=" & varValue & Chr(0)
    Next varKey

    If WritePrivateProfileSection(strSectionName, strSection, mstrFN) = 0 Then
        Call MsgBox("Ini file write section call unsuccessful", _
        vbExclamation, "Error")
    End If

End Sub

' returns the number of keys currently in the key/value collections
Public Property Get NumKeys() As Integer

    NumKeys = mcolKeys.Count

End Property

'gets a value
Public Function GetProfileString(ByVal strSectionName As String, _
    ByVal strKeyName As String) As String

    Dim strRet As String
    strRet = Space(5000)

    Call GetPrivateProfileString(strSectionName, strKeyName, "Not Found", _
        strRet, 5000, mstrFN)
    GetProfileString = TrimAtFirstNull(strRet)

End Function

'sets a value
Public Sub WriteProfileString(ByVal strSectionName As String, _
    ByVal strKeyName As String, ByVal strKeyValue As String)

    Call WritePrivateProfileString(strSectionName, strKeyName, strKeyValue, _
        mstrFN)

End Sub
```

```
' LockedItem.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1 'True
END
Attribute VB_Name = "LockedItem"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mstrLockedFN As String

Private mudtWord As MSWord

Private mdocLockedItem As Document

Private mudtItemType As ItemType

Private mudtDeliveryMode As DeliveryMode

Public Enum DeliveryMode
   dmCBT = 0
   dmPPT = 1
End Enum

Public Property Let LockedItemFileName(ByVal strNewValue As String)

   mstrLockedFN = strNewValue

End Property

Public Property Let WordInstance(ByVal udtNewValue As MSWord)

   Set mudtWord = udtNewValue

End Property

Public Property Get DeliveryMode() As DeliveryMode

   DeliveryMode = mudtDeliveryMode

End Property
```

```
Public Property Get ItemType() As ItemType

   ItemType = mudtItemType

End Property

Public Function OpenLockedItemDoc() As Boolean

   Dim udtProgress As New Progress

   Call udtProgress.Init(2, "Opening locked item...")
   udtProgress.Advance

   Set mdocLockedItem = mudtWord.WordApp.Documents.Open(mstrLockedFN)

   If mdocLockedItem.ProtectionType <> wdNoProtection Then
      Call mdocLockedItem.Unprotect("ItemEdit")
   End If

   OpenLockedItemDoc = AnalyzeLockedItem

   udtProgress.Advance

End Function

Public Sub CloseLockedItemDoc()

   mdocLockedItem.Close

   Clipboard.Clear

End Sub

Private Function AnalyzeLockedItem() As Boolean

   ' true if document is successfully analyzed
   AnalyzeLockedItem = True

   If mdocLockedItem.Tables.Count = 1 Then ' QC item
      mudtItemType = ptQuantComp
      If mdocLockedItem.Bookmarks.Count = 3 Then
         mudtDeliveryMode = dmPPT
     Else
         mudtDeliveryMode = dmCBT
     End If
```

```
        ElseIf mdocLockedItem.ListParagraphs.Count = 2 Then ' DS
            mudtItemType = ptDataSuff
            mudtDeliveryMode = dmCBT

        ElseIf mdocLockedItem.ListParagraphs.Count = 5 Then ' SMC
            mudtItemType = ptStandardMC
            mudtDeliveryMode = dmCBT

        ElseIf mdocLockedItem.Bookmarks.Exists("prop_key") = True Then 'SMC
            mudtItemType = ptStandardMC
            mudtDeliveryMode = dmPPT

        Else
          AnalyzeLockedItem = False
        End If

    End Function

    Public Sub ConvertCBTSMCItem()

        Dim udtProgress As New Progress

        Call udtProgress.Init(2, "Converting SMC CBT locked item...")

        Dim tcaDoc As Document
        Set tcaDoc = mudtWord.WordApp.ActiveDocument

        Dim stemRange As Range
        Set stemRange = mdocLockedItem.Content
        stemRange.Find.Style = "Heading 2"
        stemRange.Find.Execute FindText:="Stem"
        stemRange.Start = stemRange.Start + 5

        Dim respRange As Range
        Set respRange = mdocLockedItem.Content
        respRange.Find.Style = "Heading 2"
        respRange.Find.Execute FindText:="Response"

        stemRange.End = respRange.Start - 1
        stemRange.Copy

        Dim destRange As Range
        Set destRange = tcaDoc.Bookmarks("stem1").Range

        With destRange
```

```
'    .Borders.Enable = False
    .Words(1).Delete Count:=6
    .Collapse
    .Paste
    .Style = wdStyleNormal
'    .Borders.Enable = True
  End With

'  destRange.Borders.Enable = False
'  destRange.Collapse
'  destRange.Delete
'  destRange.Paste
'  destRange.InsertParagraphAfter
'  destRange.Style = wdStyleNormal
'  destRange.Borders.Enable = True

  With destRange.ParagraphFormat.Borders
   .Enable = True
   .DistanceFromTop = 1
   .DistanceFromLeft = 4
   .DistanceFromBottom = 1
   .DistanceFromRight = 4
  End With

  If destRange.Borders.InsideLineStyle = True Then
    destRange.Borders.InsideLineStyle = wdLineStyleNone
  End If

'  tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

  Dim nextRange As Range
  Dim Key As String
  Dim abcde As String
  abcde = "ABCDE"
  Dim i As Integer
  Dim n As Integer
  n = 1

  Dim udtIF As New IniFile
  udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
  Key = udtIF.GetProfileString("LockedItemData", "Key")

  udtProgress.Advance

  Dim tabchr As String
```

```
    tabchr = Chr$(9)

    For i = 1 To 5

      Set respRange = mdocLockedItem.ListParagraphs(i).Range
      respRange.Copy

      If Key = Mid(abcde, i, 1) Then
        Set destRange = tcaDoc.Bookmarks("Key").Range
      Else
        Set destRange = tcaDoc.Bookmarks("resp" & Format(n)).Range
        n = n + 1
      End If

      With destRange
        .Borders.Enable = False
        .Words(1).Delete
        .Collapse
        .Paste
        .Style = wdStyleNormal
        .Borders.Enable = True

        If .Words(1).Text = tabchr Then
          .Words(1).Delete
        End If

        .Words(destRange.Words.Count).Delete
      End With

    Next

    udtProgress.Advance

End Sub

Public Sub ConvertPPTSMCItem()

    Dim udtProgress As New Progress

    Call udtProgress.Init(2, "Converting SMC PPT locked item...")

    Dim tcaDoc As Document
    Set tcaDoc = mudtWord.WordApp.ActiveDocument

    Dim stemStart As Long
    Dim destRange As Range
    Set destRange = tcaDoc.Bookmarks("stem1").Range
    stemStart = destRange.Start

    Dim stemRange As Range
'   Set stemRange = mdocLockedItem.Bookmarks("itemnum").Range
'   stemRange.Start = stemRange.Start + 1
    Set stemRange = mdocLockedItem.Content
    stemRange.Find.Style = "PPTStimulus"

    If stemRange.Find.Execute Then
       stemRange.Copy
       destRange.Paste
       destRange.Collapse Direction:=wdCollapseEnd
    End If

    Set stemRange = mdocLockedItem.Content
    stemRange.Find.Style = "PPTStem"
    stemRange.Find.Execute
    stemRange.Copy
    destRange.Paste
    destRange.Style = wdStyleNormal

    destRange.Start = stemStart
    destRange.Borders.Enable = True

'    With destRange.ParagraphFormat.Borders
'       .Enable = True
'       .DistanceFromTop = 1
'       .DistanceFromLeft = 4
'       .DistanceFromBottom = 1
'       .DistanceFromRight = 4
'    End With

    If destRange.Borders.InsideLineStyle = True Then
      destRange.Borders.InsideLineStyle = wdLineStyleNone
    End If

    tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

    Dim nextRange As Range
    Dim respRange As Range
    Dim Key As String
    Dim abcde As String
    abcde = "ABCDE"
```

```
Dim i As Integer
Dim n As Integer
n = 1

Dim udtIF As New IniFile
udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
Key = udtIF.GetProfileString("LockedItemData", "Key")

udtProgress.Advance

For i = 1 To 5

  Set respRange = mdocLockedItem.Content
  respRange.Find.Style = "PPTOptions"
  respRange.Find.Execute FindText:="(" & Mid(abcde, i, 1) & ")"
  respRange.Start = respRange.Start + 4

  Set nextRange = mdocLockedItem.Content

  If i < 5 Then
     nextRange.Find.Style = "PPTOptions"
     nextRange.Find.Execute FindText:="(" & Mid(abcde, i + 1, 1) & ")"
  Else
     nextRange.Find.Style = "ItemLabel"
     nextRange.Find.Execute FindText:="Scratch Pad"
  End If

  respRange.End = nextRange.Start - 1
  respRange.Copy

  If Key = Mid(abcde, i, 1) Then
     Set destRange = tcaDoc.Bookmarks("Key").Range
  Else
     Set destRange = tcaDoc.Bookmarks("resp" & Format(n)).Range
     n = n + 1
  End If

  destRange.Words(1).Delete
  destRange.Collapse
  destRange.Paste

Next

udtProgress.Advance
```

```
End Sub

Public Sub ConvertDSItem()

  Dim udtProgress As New Progress

  Call udtProgress.Init(2, "Converting DS CBT locked item...")

  Dim tcaDoc As Document
  Set tcaDoc = mudtWord.WordApp.ActiveDocument

  Dim stemRange As Range
  Set stemRange = mdocLockedItem.Content
  stemRange.Find.Style = "Heading 2"
  stemRange.Find.Execute FindText:="Stem"
  stemRange.Start = stemRange.Start + 5

  Dim respRange As Range
  Set respRange = mdocLockedItem.Content
  respRange.Find.Style = "DataSuffStatement"
  respRange.Find.Execute

  stemRange.End = respRange.Start - 1
  stemRange.Copy

  Dim destRange As Range
  Set destRange = tcaDoc.Bookmarks("stem1").Range
  destRange.Borders.Enable = False
  destRange.Collapse
  destRange.Paste
'  destRange.Borders.Enable = True

  With destRange.ParagraphFormat.Borders
   .Enable = True
   .DistanceFromTop = 1
   .DistanceFromLeft = 4
   .DistanceFromBottom = 1
   .DistanceFromRight = 4
  End With

  If destRange.Borders.HasHorizontal = True Then
    destRange.Borders(wdBorderHorizontal).LineStyle = wdLineStyleNone
  End If

  Dim Key As String
```

```
    Dim udtIF As New IniFile
    udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
    Key = udtIF.GetProfileString("LockedItemData", "Key")

    Set destRange = tcaDoc.Bookmarks("Key").Range
    destRange.Words(1).Delete
    destRange.InsertBefore Text:=Key

    udtProgress.Advance

    Dim i As Integer

    For i = 1 To 2

      Set respRange = mdocLockedItem.ListParagraphs(i).Range
      respRange.Copy

      Set destRange = tcaDoc.Tables(i).Cell(Row:=1, Column:=1).Range
      destRange.Paste
      destRange.Style = wdStyleNormal

    Next

    udtProgress.Advance

End Sub

Public Sub ConvertCBTQCItem()

    Dim udtProgress As New Progress

    Call udtProgress.Init(2, "Converting QC CBT locked item...")

    Dim tcaDoc As Document
    Set tcaDoc = mudtWord.WordApp.ActiveDocument

    Dim stemRange As Range
    Set stemRange = mdocLockedItem.Tables(1).Cell(Row:=1, Column:=1).Range
    stemRange.Copy

    Dim destRange As Range
    Set destRange = tcaDoc.Bookmarks("stem1").Range
    destRange.Borders.Enable = False
    destRange.Words(2).Delete
    destRange.Words(1).Delete
    destRange.Collapse
    destRange.Paste
    tcaDoc.Tables(2).Rows.SetLeftIndent LeftIndent:=-0.6, RulerStyle:=wdAdjustNone
    tcaDoc.Tables(2).ConvertToText Separator:=wdSeparateByTabs
    destRange.Borders.Enable = True
    tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

    Dim Key As String
    Dim udtIF As New IniFile
    udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
    Key = udtIF.GetProfileString("LockedItemData", "Key")

    Set destRange = tcaDoc.Bookmarks("Key").Range
    destRange.Words(1).Delete
    destRange.InsertBefore Text:=Key

    udtProgress.Advance

    Dim respRange As Range
    Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=1).Range
    respRange.Copy
    Set destRange = tcaDoc.Bookmarks("columnA").Range
    destRange.Collapse
    destRange.Paste

    Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=2).Range
    respRange.Copy
    Set destRange = tcaDoc.Bookmarks("columnB").Range
    destRange.Collapse
    destRange.Paste

    udtProgress.Advance

End Sub

Public Sub ConvertPPTQCItem()

    Dim udtProgress As New Progress

    Call udtProgress.Init(2, "Converting QC PPT locked item...")

    Dim tcaDoc As Document
    Set tcaDoc = mudtWord.WordApp.ActiveDocument

    Dim stemRange As Range
    Set stemRange = mdocLockedItem.Tables(1).Cell(Row:=1, Column:=2).Range
    stemRange.Copy

    Dim destRange As Range
    Set destRange = tcaDoc.Bookmarks("stem1").Range
    destRange.Borders.Enable = False
    destRange.Words(2).Delete
    destRange.Words(1).Delete
    destRange.Collapse
    destRange.Paste
    tcaDoc.Tables(2).Rows.SetLeftIndent LeftIndent:=-0.6, RulerStyle:=wdAdjustNone
    tcaDoc.Tables(2).ConvertToText Separator:=wdSeparateByTabs
    destRange.Borders.Enable = True
    tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

    Dim Key As String
    Dim udtIF As New IniFile
    udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
    Key = udtIF.GetProfileString("LockedItemData", "Key")

    Set destRange = tcaDoc.Bookmarks("Key").Range
    destRange.Words(1).Delete
    destRange.InsertBefore Text:=Key

    udtProgress.Advance

    Dim respRange As Range
    Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=2).Range
    respRange.Copy
    Set destRange = tcaDoc.Bookmarks("columnA").Range
    destRange.Collapse
    destRange.Paste

    Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=4).Range
    respRange.Copy
    Set destRange = tcaDoc.Bookmarks("columnB").Range
    destRange.Collapse
    destRange.Paste

    udtProgress.Advance

End Sub
```

09654949.090100

```
' Model.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Model"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"No"
Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

' enable i/o
Private mudtFile As File

' handle for Model
Private mdocModel As Document

' the .doc file name of this model
Private mstrDocFN As String

' the .mdl file name of this model
Private mstrConFN As String

' has this model produced variants that were accepted?
Private mblnIsFrozen As Boolean

' comments about this model
Private mstrComments As String

' all of the variables for this model
Private mudtCVariables As CVariables

' all of the constraints for this model
Private mudtCConstraints As CConstraints
```

```
' all of the clones generated by this model
Private mudtCClones As CClones

' the collection of checksums accepted by this model (these persist)
Private mcolChecksums As Collection

' the collection of checksums accepted by this model (these don't persist)
Private mcolTempChecksums As Collection

' the Prolog object
Private mudtProlog As Prolog

' needed for I/O
Private mblnProcessChecksums As Boolean

' is dirty?
Private mblnIsDirty As Boolean

' needed to save the model one last time after it's frozen
Private mblnFreeze As Boolean

Private Enum ModelRecordLayout
    mrLocalDataIndex = 1 ' long (takes 4 bytes)
    mrVariableIndex = 5 ' long
    mrConstraintIndex = 9 ' long
    mrChecksumIndex = 13 ' ' long
    mrLocalData = 51 ' byte
    mrVariables = 201 ' variable length
    ' the constraint data starts wherever the checksum data ends
    ' the checksum data starts wherever the constraint data ends
End Enum

Private Sub Class_Initialize()

    Set mudtCVariables = New CVariables
    Set mudtCConstraints = New CConstraints
    Set mudtCClones = New CClones
    Set mcolChecksums = New Collection
    Set mcolTempChecksums = New Collection
    mblnIsDirty = True
    mblnFreeze = False

End Sub

Public Property Get FileName() As String
```

```
    FileName = mstrDocFN

End Property

Public Property Let FileName(ByVal strNewValue As String)

    mstrDocFN = strNewValue

    ' create the FN for the constraint file
    mstrConFN = left(mstrDocFN, Len(mstrDocFN) - 4) & ".mdl"

End Property

Public Property Get IsFrozen() As Boolean

    IsFrozen = mblnIsFrozen

End Property

Public Property Let IsFrozen(ByVal blnNewValue As Boolean)

    mblnIsFrozen = blnNewValue

End Property

Public Property Get Comments() As String

    Comments = mstrComments

End Property

Public Property Let Comments(ByVal strNewValue As String)

    If mstrComments <> strNewValue Then
        mstrComments = strNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get Clones() As CClones

    Set Clones = mudtCClones

End Property
```

```
Public Property Get Variables() As CVariables

   Set Variables = mudtCVariables

End Property

Public Property Get Constraints() As CConstraints

   Set Constraints = mudtCConstraints

End Property

Public Sub FreezeModel()

   If IsFrozen = False Then
      mblnFreeze = True
      IsFrozen = True
      WriteModel
   End If

End Sub

Public Sub AddChecksum(ByVal dblChecksum As Double)

   Call mcolChecksums.Add(dblChecksum)
   mblnIsDirty = True

End Sub

' resets the checksums if this model is a child
Public Sub InitChecksums()

   Set mcolChecksums = New Collection

End Sub

Private Sub AddTempChecksum(ByVal dblChecksum As Double)

   Call mcolTempChecksums.Add(dblChecksum)

End Sub

' resets the temp checksums if this model is changed and variants are deleted
Public Sub InitTempChecksums()
```

```
    Set mcolTempChecksums = New Collection

End Sub

Public Function ChecksumExists(ByVal dblChecksum As Double) As Boolean

    Dim vntChecksum As Variant

    ' if no variables were checksummed, consider the variant unique
    If dblChecksum = 0 Then
        ChecksumExists = False
        Exit Function
    End If

    ' check the persistent checksums (from accepted or discarded variants)
    For Each vntChecksum In mcolChecksums
        If vntChecksum = dblChecksum Then
            ChecksumExists = True
            Exit Function
        End If
    Next vntChecksum

    ' check the checksums of variants produced in this session
    For Each vntChecksum In mcolTempChecksums
        If vntChecksum = dblChecksum Then
            ChecksumExists = True
            Exit Function
        End If
    Next vntChecksum

   ChecksumExists = False

End Function

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

    mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

    Dim mblnSaved As Boolean

    ' As frozen models never get saved, they report is dirty
    ' when they are read in from disk.  This fix causes them
    ' to always report not IsDirty.
'   If IsFrozen Then
'       IsDirty = False
'       Exit Property
'   End If

    If mdocModel Is Nothing Then
        mblnSaved = True
    Else
        mblnSaved = mdocModel.Saved
    End If

    If mblnIsDirty Or _
        mudtCVariables.IsDirty Or _
        mudtCConstraints.IsDirty Or _
        mblnSaved = False Then
            IsDirty = True
    Else
        IsDirty = False
    End If

End Property

Public Property Let LastClone(ByVal intNewValue As Integer)

    mudtCClones.SeqNum = intNewValue

End Property

Public Property Get LastClone() As Integer

    LastClone = mudtCClones.SeqNum

End Property

' displays model
Public Sub OpenDoc(ByVal udtWord As MSWord)

    Dim udtDS As New DocStatus

    ' see if word doc is open
    If udtDS.IsOpen(mstrDocFN) = False Then
        Set mdocModel = udtWord.WordApp.Documents.Open(mstrDocFN, , mblnIsFrozen)
    End If
    mdocModel.Activate

End Sub

' closes model
Public Sub CloseDoc()

    ' save the model and the word doc
    Call WriteModel

    Dim udtDS As New DocStatus

    ' close the word doc
    If udtDS.IsOpen(mstrDocFN) Then
        Call mdocModel.Close(False) ' don't save
        Set mdocModel = Nothing
    End If

End Sub

Public Sub CloseAllCloneDocs()

    Dim udtClone As Clone

    For Each udtClone In mudtCClones
        udtClone.CloseDoc
    Next udtClone

End Sub

Public Sub ReadModel()

    Dim udtWAPI As New Win32API

    If udtWAPI.FileExists(mstrConFN) Then
        Set mudtFile = New File
        mudtFile.FileName = mstrConFN
        mblnProcessChecksums = False
        Call mudtFile.ReadFile(Me, mrLocalDataIndex, mrVariableIndex)
        Call mudtCVariables.ReadCollection(mstrConFN, mrVariableIndex, mrConstraintIndex)
        Call mudtCConstraints.ReadCollection(mstrConFN, mrConstraintIndex,
mrChecksumIndex)
        mblnProcessChecksums = True
        Call mudtFile.ReadFile(Me, mrChecksumIndex, READ_UNTIL_EOF)
        Set mudtFile = Nothing
    End If

End Sub

Public Sub ReadObjects()

    Dim vField As Variant

    If mblnProcessChecksums Then
        On Error GoTo BeatIt
        Do Until err.Number <> 0
            Call mudtFile.ReadField(vField)
            Call mcolChecksums.Add(vField)
        Loop
    Else
        Call mudtFile.ReadField(vField) ' returns the version stamp
        Call mudtFile.ReadField(vField)
        LastClone = vField
        Call mudtFile.ReadField(vField)
        IsFrozen = vField
        Call mudtFile.ReadField(vField)
        Comments = vField
    End If

BeatIt:

    Exit Sub

End Sub

Public Sub WriteModel()

    Dim lngEndPos As Long
    Dim udtDS As New DocStatus
    Dim udtProg As New Progress

    If IsDirty = False Then Exit Sub

'   If IsFrozen And mblnFreeze = False Then Exit Sub

    Call udtProg.Init(2, "Saving the active model...")
    If udtDS.IsOpen(mstrDocFN) Then ' see if word doc is open
        If Not IsFrozen Then ' command will fail if doc is read-only
            mdocModel.Save
        End If
    End If
    Set mudtFile = New File
    mudtFile.FileName = mstrConFN
    mblnProcessChecksums = False
    Call mudtFile.WriteFile(Me, True, mrLocalDataIndex, mrLocalData)
    udtProg.Advance
    lngEndPos = mudtCVariables.WriteCollection(mstrConFN, mrVariableIndex, mrVariables)
    lngEndPos = mudtCConstraints.WriteCollection(mstrConFN, mrConstraintIndex, lngEndPos)
    mblnProcessChecksums = True
    Call mudtFile.WriteFile(Me, False, mrChecksumIndex, lngEndPos)
    Set mudtFile = Nothing
    udtProg.Advance
    IsDirty = False
    mblnFreeze = False

End Sub

Public Sub WriteObjects()

    Dim vntChecksum As Variant

    If mblnProcessChecksums Then
        For Each vntChecksum In mcolChecksums
            Call mudtFile.WriteField(vntChecksum)
        Next vntChecksum
    Else
        Call mudtFile.WriteField(mintVERSIONSTAMP)
        Call mudtFile.WriteField(LastClone)
        Call mudtFile.WriteField(IsFrozen)
        Call mudtFile.WriteField(Comments)
    End If

End Sub

' tests the constraints, doesn't care about unique solution
Public Function ConstraintsOK(ByVal udtTestType As TestType, _
    ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
    blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

    Dim strVN As String
    Dim strVal As String

    Dim udtCS As ConstraintSolver

    Set udtCS = InitConstraintSolver(2, udtTestType)
    udtCS.Prolog = udtProlog

    blnUnderconstrained = False
    blnTestAborted = False

    Select Case udtCS.Solve(srTest)
      Case srPrologError, srNoSolutions
        ConstraintsOK = False
        Exit Function
      Case srPrologAborted
        blnTestAborted = True
        ConstraintsOK = False
        Exit Function
      Case srSuccess
        Do While udtCS.GetNextValue(strUnderconstrainedVN, strVal)
          If strVal = "_" Then ' it's underconstrained
            ConstraintsOK = False
            blnUnderconstrained = True
            Exit Function
          End If
        Loop
    End Select

    ConstraintsOK = True

End Function

' implemented in the subclasses of Model
Public Sub GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
    ByVal intNumClones As Integer, ByVal bytDifference As Byte)

End Sub

' common code called by GenerateClones in the subclasses
Public Sub SubstituteValues(ByVal objO As Object, _
    ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
    ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
    ByVal intStartPos As Integer)

    Dim udtClone As Clone
    Dim strPath As String
    Dim fRange As Range
    Dim intIndex As Integer
    Dim udtCS As ConstraintSolver
Dim udtSortedVs As CVariables
Dim udtCon As Constraint
Dim strVarName As String
Dim strValue As String
Dim intTry As Integer
Dim blnSolFound As Boolean
Dim blnUniqueSolFound As Boolean
Dim udtType As VariableType

CloseDoc ' close the model doc
CommandBars("File").Controls("Exit").Enabled = False
Randomize

' do substitution of values into model doc

strPath = ExtractPath(FileName)

'   Dim udtProgress As New Progress
'   Call udtProgress.Init(intNumClones, "Generating variants...")

' initalize the constraint solver
Set udtCS = InitConstraintSolver(bytDifference)
udtCS.Prolog = udtProlog

' solve loop
For intIndex = 1 To intNumClones
   ' try 10x to get a unique sol, then give up
   For intTry = 1 To 10
      DoEvents ' allow abort
      If frmProlog.Abort Then
         Exit Sub
      End If
      blnSolFound = False
      blnUniqueSolFound = False
      If udtCS.Solve(srGenerate) Then  ' found a variant
         blnSolFound = True
      Else
         Exit For
      End If
      ' variant found - is it unique?
      If Not ChecksumExists(udtCS.Checksum) Then
         blnUniqueSolFound = True
         Exit For
      End If
   Next intTry
```

```
    ' error if no solution found
    If Not blnSolFound Then
       Call MsgBox("No solution could be found for this constraint set", _
          vbExclamation, "Error")
'          udtProgress.Kill
       Exit Sub
    End If
    ' error if unique solution could not be found
    If Not blnUniqueSolFound Then
       Call MsgBox("A unique solution could not be found for this constraint set after 10
attempts." & _
          "  You may want to try again.", vbExclamation, "Error")
'          udtProgress.Kill
       Exit Sub
    End If
    ' add the new clone to the collection
    Set udtClone = Clones.Add(ExtractFileName(FileName), True)
    udtClone.Checksum = udtCS.Checksum
    Call AddTempChecksum(udtClone.Checksum)
    ' add the new clone to the disposition list box
    With frmTCA.lstDisposition
       Call .AddItem(udtClone.FileName)
       .ItemData(.ListCount - 1) = udtClone.index
    End With
    FileCopy FileName, strPath & udtClone.FileName
    Call udtClone.OpenDoc(udtWord, strPath)
    ' do the substitution
    Set fRange = udtClone.CloneDoc.Content
    fRange.start = intStartPos

    With fRange.find
       While udtCS.GetNextValue(strVarName, strValue)
          .ClearFormatting
          .Text = strVarName
          .Replacement.ClearFormatting
          .Replacement.Text = FormatValue(strVarName, strValue)
          ' this first execute needed so Word returns correct value
          .Execute replace:=wdReplaceAll, Forward:=True, _
             MatchCase:=True
       Wend
    End With

    Dim i, n As Integer
    Dim nShapes As Long
```

```
'       n = udtClone.CloneDoc.InlineShapes.Count
'
'       For i = 1 To n
'          udtCS.ResetValueIndex
'
'          While udtCS.GetNextValue(strVarName, strValue)
'             udtClone.CloneDoc.InlineShapes(i).Select
'             Call MTTextSubstitution(strVarName, strValue)
'          Wend
'       Next

        udtClone.CloneDoc.Bookmarks("stem1").Range.Copy

        If udtClone.CloneDoc.Bookmarks.Exists("tca_Stem") = True Then
           Dim stemRange As Range
           Set stemRange = udtClone.CloneDoc.Bookmarks("tca_Stem").Range
           stemRange.Paste
           udtClone.CloneDoc.Bookmarks.Add name:="tca_Stem", Range:=stemRange
        Else
           Call MsgBox("Model is missing TCA_Stem Bookmark!", vbExclamation, "Hey!")
        End If

        ' trim hard returns at end of stem
        Dim retchr As String
        retchr = Chr$(13)

        With stemRange
           n = 0
           i = .Words.Count

           While .Words(i).Text = retchr And i > 1 ' Rob: I added the And part. Pete
             i = i - 1
             If .Words(i).Text = retchr Then
               n = n + 1
             End If
           Wend

           If n > 0 Then
             .Words(.Words.Count - n + 1).Delete Count:=n
           End If
        End With

        ' callback to subclass to code unique to this model type
        Call objO.CreateVariant(udtClone)
'       udtProgress.Advance
```

```
    udtClone.CloseDoc
  Next intIndex

End Sub

' create, initialize constraint solver
Private Function InitConstraintSolver(ByVal bytDifference As Byte, _
  Optional ByVal udtTestType As TestType = tcTestAll) As ConstraintSolver

  Dim udtVar As Variable
  Dim udtCon As Constraint
  Dim udtVarString As VarString
  Dim udtCS As New ConstraintSolver
  Dim udtSortedVs As CVariables

  ' add enabled variables to ConstraintSolver object, sorted by length,
  ' strings first

  Set udtSortedVs = mudtCVariables.SortVarNamesByLength

  For Each udtVar In udtSortedVs
    If udtVar.Enabled Then
      Call udtCS.AddVariable(udtVar)
    End If
  Next udtVar

  ' Add enabled constraints
  For Each udtCon In Constraints
    If udtCon.Enabled Then
      If udtTestType = tcTestAll Or _
        udtCon.ConstraintType = udtTestType - 1 Then
        Call udtCS.AddConstraint(udtCon)
      End If
    End If
  Next udtCon

  udtCS.DiffWeight = bytDifference

  Set InitConstraintSolver = udtCS

End Function

' formats all math variables for item presentation
Private Function FormatValue(ByVal strVarName As String, _
  ByVal strValue As String) As String
```

```
    Dim udtV As Variable
    Dim udtVR As VarReal
    Dim udtVF As VarFraction

    For Each udtV In mudtCVariables
        If udtV.Enabled Then
            If udtV.name = ExtractVarName(strVarName) Then
                Select Case udtV.Typ
                    Case vtInteger
                        FormatValue = strValue
                    Case vtReal
                        Set udtVR = udtV
                        FormatValue = FormatReal(strValue, _
                            udtVR.DecimalPlaces, udtVR.TrailingZeros)
                    Case vtFraction
                        Set udtVF = udtV
                        If udtVF.MixedNumbers Then
                            FormatValue = FormatFraction(strValue)
                        Else
                            FormatValue = strValue
                        End If
                    Case vtString
                        FormatValue = strValue
                    Case vtUntyped
                        FormatValue = FormatUntyped(strValue)
                End Select
                Exit For
            End If
        End If
    Next udtV

End Function

' takes the index off of a string variable name that is indexed
Private Function ExtractVarName(ByVal strName As String) As String

    Dim varI As Variant

    varI = InStr(1, strName, ".")

    If varI > 0 Then
        ExtractVarName = left(strName, varI - 1)
    Else
        ExtractVarName = strName
    End If
End Function

' formats reals for item presentation
Private Function FormatReal(ByVal strReal As String, ByVal intPlaces As Integer, _
   ByVal blnTZeros As Boolean) As String

   Dim varPos As Variant
   Dim intLen As Integer
   Dim strI As String
   Dim strD As String
   Dim blnZeroFound As Boolean

   varPos = InStr(1, strReal, ".")

   ' isolate strings on either side of decimal point
   If varPos = 0 Then
      strI = strReal
   Else
      strI = Mid(strReal, 1, varPos - 1)
      strD = Mid(strReal, varPos + 1, Len(strReal))
   End If

   intLen = Len(strD)

   ' pad or trim to intPlaces
   If intLen < intPlaces Then
      strD = strD & String(intPlaces - intLen, "0")
   Else
      If intLen > intPlaces Then
         strD = left(strD, intPlaces)
      End If
   End If

   ' get rid of trailing zeros if desired
   If blnTZeros = False Then
      Do
         blnZeroFound = False
         If right(strD, 1) = "0" Then
            strD = left(strD, Len(strD) - 1)
            blnZeroFound = True
         End If
      Loop While blnZeroFound
   End If

   ' reassemble string
```

```
    If Len(strD) > 0 Then
      FormatReal = strI & "." & strD
    Else
      FormatReal = strI
    End If

End Function

' formats fraction as mixed number for item presentation
Private Function FormatFraction(ByVal strFraction As String) As String

    Dim intNum As Integer
    Dim intDen As Integer
    Dim intQuot As Integer
    Dim vntI As Variant

    vntI = InStr(strFraction, "/")

    ' it's an integer
    If vntI = 0 Then ' it's a whole number
      FormatFraction = strFraction
      Exit Function
    End If

    intNum = CInt(left(strFraction, vntI - 1))
    intDen = CInt(right(strFraction, Len(strFraction) - vntI))

    If intDen > 0 And Abs(intNum) > intDen Then
      intQuot = Int(intNum / intDen)
      intNum = intNum Mod intDen
      FormatFraction = Trim(Str(intQuot)) & " " & Trim(Str(Abs(intNum))) & "/" & _
        Trim(Str(intDen))
    Else
      FormatFraction = strFraction
    End If

End Function

Private Function FormatUntyped(ByVal strValue As String)

    Dim varI As Variant

    ' see if the value is a list - if so, it will be in []
    If left(strValue, 1) = "[" And right(strValue, 1) = "]" Then
      ' trim the brackets off
    FormatUntyped = Mid(strValue, 2, Len(strValue) - 2)
  Else
    FormatUntyped = strValue
  End If

End Function

Private Function MTTextSubstitution(Source As String, dest As String)
  Dim stat

  Selection.Copy

  'Init API, reset transform
  If MTUtil.CheckMTDLLVersion = 0 Then Exit Function
  MTXFormReset

  'first substitution
  stat = MTXFormAddVarSub( _
    mtxfmSUBST_ALL, _
    mtxfmVAR_SUB_PLAIN_TEXT, Source, 0, _
    mtxfmVAR_SUB_PLAIN_TEXT, dest, Len(dest), mtxfmSTYLE_NUMBER)

  If stat <> 0 Then
    MsgBox "MTXFormAddVarSub returned: " + Str(stat)
    Exit Function
  End If

  'do the substitution
  stat = TransformGraphicEquation
  If stat <> 0 Then
    MsgBox "TransformGraphicEquation returned: " + Str(stat)
    Exit Function
  End If

  MTTermAPI

  Selection.Delete

  'Paste new equation
  Selection.Collapse Direction:=wdCollapseEnd
  Selection.PasteSpecial Placement:=wdInLine

End Function
```

09654949.090100

```
' PrintModel.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "PrintModel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mstrModelName As String
Private mstrNow As String
Private mintPage As Integer
Private mintTab As Integer

Public Property Let ModelName(ByVal strNewValue As String)

   mstrModelName = strNewValue

End Property

Public Sub PrintString(ByVal strS As String, ByVal intIndent As Integer)

   CheckPageBreak

   If Printer.CurrentY = 0 Then PrintHeading

   Printer.Print Space(intIndent * mintTab) & strS

End Sub

Private Sub PrintHeading()

   Dim intY As Integer

   Printer.CurrentY = 1440 ' top margin
   Printer.Print Space(mintTab) & _
      "Variables and constraints for model " & mstrModelName
   Printer.Print Space(mintTab) & mstrNow
   Printer.CurrentY = Printer.CurrentY + 100
   Printer.Line Step(0, 0)-Step(Printer.Width, 0)
   SkipLine
```

```
    intY = Printer.CurrentY
    Printer.CurrentY = Printer.Height - 1700
    Printer.Line Step(0, 0)-Step(Printer.Width, 0)
    Printer.CurrentY = Printer.CurrentY + 100
    Printer.CurrentX = 0
    Printer.Print Space(mintTab) & "Page " & Str(mintPage)
    Printer.CurrentY = intY
    mintPage = mintPage + 1

End Sub

Private Sub SkipLine()

    Printer.Print " "

End Sub

Private Sub CheckPageBreak()

    Select Case Printer.PaperSize
        Case vbPRPSLetter, vbPRPSLetterSmall
            Call CheckOrientation(8.5, 11)
        Case vbPRPSTabloid
            Call CheckOrientation(11, 17)
        Case vbPRPSLedger
            Call CheckOrientation(17, 11)
        Case vbPRPSLegal
            Call CheckOrientation(8.5, 14)
    End Select

End Sub

Private Sub CheckOrientation(ByVal sngWidth As Single, _
    ByVal sngHeight As Single)

    ' convert inches to twips
    sngWidth = sngWidth * 1440
    sngHeight = sngHeight * 1440

    If Printer.Orientation = vbPRORPortrait Then
        If Printer.CurrentY >= sngHeight - 2200 Then
            Printer.NewPage
        End If
    Else
        If Printer.CurrentY >= sngWidth - 2200 Then
```

```
            Printer.NewPage
        End If
    End If

End Sub

Private Sub Class_Initialize()

    Printer.FontSize = 11
    mstrNow = Now
    mintPage = 1
    mintTab = 4

End Sub

Private Sub Class_Terminate()

    Printer.EndDoc

End Sub
```

```
' Progress.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "Progress"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
' class to give visual indication of progress
Option Explicit

Private mintStepSize As Integer

' pulls up form
Public Sub Init(ByVal intNumIncrements As Integer, _
   Optional ByVal strCaption As String)

   If intNumIncrements = 0 Then ' prevent divide by 0
      Beep
      Exit Sub
   End If

   mintStepSize = 500 / intNumIncrements
   frmProgress.prbProgressBar.Max = mintStepSize * intNumIncrements

   If Len(strCaption) > 0 Then
      frmProgress.lblProgress = strCaption
   End If

   frmProgress.Show
   frmProgress.Refresh

End Sub

' bumps the progress bar to the next increment.  When the progress
' bar is fully advanced, the form is unloaded.
Public Sub Advance()

   Dim intStop As Integer

   With frmProgress.prbProgressBar
      If .Value = .Max Then
```

```
            Exit Sub
        End If
        intStop = .Value + mintStepSize
        Do Until .Value = intStop
            .Value = .Value + 1
            If .Value = .Max Then
                Unload frmProgress
                Exit Sub
            End If
        Loop
    End With

End Sub

Public Sub AbsoluteAdvance(ByVal intNewValue As Integer)

    frmProgress.prbProgressBar.Value = intNewValue * mintStepSize

End Sub

Public Sub Kill()

    Unload frmProgress

End Sub
```

```
' Prolog.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = 0   'False
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Prolog"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

Private Declare Function StartProlog4Session Lib "prlghlapi.dll" _
   (ByVal strP4FN As String) As Long
Private Declare Function EndProlog4Session Lib "prlghlapi.dll" () As Long
Private Declare Function GetHLAPIVersion Lib "prlghlapi.dll" () As String
Private Declare Function VBGetHLAPIVersion Lib "prlghlapi.dll" () As String
Private Declare Function SolveConstraintOrdered Lib "prlghlapi.dll" _
   (ByVal Constraint As String, ByVal SolutionOrder As Long) As Long
Private Declare Function SolveConstraintRandomly Lib "prlghlapi.dll" _
   (ByVal Constraint As String) As Long
Private Declare Function SolveConstraintOrderedNSolns Lib "prlghlapi.dll" _
   (ByVal Constraint As String, ByVal SolutionOrder As Long, _
   ByVal NumSols As Long) As Long
Private Declare Function IsFullyConstrained Lib "prlghlapi.dll" _
   (ByVal Constraint As String) As Long
Private Declare Function GetValue Lib "prlghlapi.dll" _
   (ByVal strVarName As String) As Long
Private Declare Function VBGetValue_string Lib "prlghlapi.dll" _
   (ByVal udtPtr As Any) As String
Private Declare Function VBPrintAllVarVals Lib "prlghlapi.dll" () As String
Private Declare Function SetSolnDiffWt Lib "prlghlapi.dll" _
   (ByVal Weight As Long) As Long
Private Declare Function SetPrologInterruptFile Lib "prlghlapi.dll" _
   (ByVal strFN As String) As Long

'Keep the constants in sync with appropriate values in prlghlapi.h
' Solution-Orders:
```

```
Private Enum PrologOrder
   prDontCareOrder = 0
   prDifferentOrder = 10
   prLikeOrder = 20
   prRandomOrder = 30
   prUniqueOrder = 40
End Enum

Private Enum PrologType
   prValUnknown = 0
   prValInteger = 10
   prValRationalFloat = 12
   prValRationalFraction = 13
   prValIrrational = 14
   prValReal = 15
   prValString = 20
   prValList = 25
   prValFunctor = 30
   prValSymbol = 35
   prValVar = 100
End Enum

Private Enum PrologErrors
   prErrInitialization = -10
   prErrIntegerraintTooLong = -15
   prErrGettingTerm = -20
   prErrMakingFunctor = -25
   prErrInvalidInterval = -30
   prErrArityTooMany = -35
   prErrParse = -40
   prErrNullTerm = -45
End Enum

' used to hold all strings for the Prolog
Private mcolVNs As Collection

Private mstrDelimit As String

Private mintNumSols As Integer

Event Finished(ByVal lngRet As Long)

Private Sub Class_Initialize()

   Set mcolVNs = New Collection
```

```
    Set gProlog = Me ' gProlog is defined in Timer.bas

    Dim lngRet As Long

    ' if this file exists, interrupt prolog processing
    lngRet = SetPrologInterruptFile("c:\halt.tca")

End Sub

Private Sub Class_Terminate()

    Set gProlog = Nothing

End Sub

Public Property Get Version() As String

    Version = GetHLAPIVersion()

End Property

' sets the degree of difference in the variants.  Range is 0 to 2.
Public Property Let DiffWeight(ByVal bytDifference As Byte)

    Call SetSolnDiffWt(CLng(bytDifference))

End Property

Public Function StartProlog() As Boolean

    ChDir App.Path ' set path to application dir for hlp4lib.p4 file
    StartProlog = CBool(StartProlog4Session("hlp4lib.p4"))

End Function

Public Function EndProlog() As Boolean

    ChDir App.Path ' set path to application dir for hlp4lib.p4 file
    EndProlog = CBool(EndProlog4Session())

End Function

Public Sub AddVariable(ByVal strS As String)
```

```
    If Len(strS) > 0 Then ' it's not an untyped variable
        Call mcolVNs.Add(strS)
        mstrDelimit = "end_var_defs,"
    End If

End Sub

Public Sub AddConstraint(ByVal strS As String)

    Call mcolVNs.Add(mstrDelimit & strS)
    mstrDelimit = ""

End Sub

Public Sub SolveConstraintsRandomly()

    SolveAsync ' in Timer.bas - must be in a standard module

End Sub

Public Sub SolveConstraintsAsync()

    Dim strS As String
    Dim lngRet As Long

    lngRet = -1 ' default to error condition

    If mcolVNs.Count > 0 Then ' there's something for Prolog to chew on
        strS = BuildString()
        ChDir App.Path ' set path to application dir for hlp4lib.p4 file
        lngRet = SolveConstraintRandomly(strS) ' call Prolog
    End If

    RaiseEvent Finished(lngRet)

    Set mcolVNs = New Collection

End Sub

Private Function RandomNumSols() As Integer

    Randomize
    RandomNumSols = 10 * Rnd - 0.5
    If RandomNumSols = 0 Then RandomNumSols = 1
```

```
End Function

Private Sub Advance(ByVal lngRet As Long)

   Dim intI As Integer

   For intI = 1 To lngRet
      NextSolution
   Next intI

End Sub

' gets the next solution, returns true if one exists, false if it doesn't
Private Function NextSolution() As Boolean

   ChDir App.Path ' set path to application dir for hlp4lib.p4 file
   NextSolution = SolveConstraintOrderedNSolns(vbNullString, _
      prUniqueOrder, mintNumSols)

End Function

Public Property Get PrintAllVals() As String

   PrintAllVals = VBPrintAllVarVals

End Property

' get the values associated with each solution
Public Property Get Value(ByVal strVN As String) As String

   Dim lngPtr As Long
   Dim strT As String

   ChDir App.Path ' set path to application dir for hlp4lib.p4 file
   lngPtr = GetValue(strVN) ' returns a pointer to the variable

   If lngPtr Then ' to handle untyped variables that have no constraint, and therefore no value
    strT = VBGetValue_string(lngPtr) ' returns a string
      Value = Left(strT, Len(strT) - 1) ' trim off the null delimiter
   Else
      Value = "_"
   End If

End Property
```

```
Private Function BuildString() As String

   Dim varStr As Variant
   Dim strS As String

   For Each varStr In mcolVNs
      strS = strS & varStr & ", "
   Next varStr

   ' trim off the last comma and space
   strS = Left(strS, Len(strS) - 2)
   ' add a period
   strS = strS & "."

   BuildString = strS

End Function

Public Sub ShowString()

   Dim strS As String

   strS = BuildString()
'   Call MsgBox(strS, , "Prolog string is:")

End Sub
```

```
' PSMODEL.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "SMCModel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Model

Dim mudtModel As Model
Dim lastStart As Integer

Private Sub Class_Initialize()

    Set mudtModel = New Model

End Sub

' Delegated to Class Model
Public Property Get Model_FileName() As String

    Model_FileName = mudtModel.FileName

End Property

' Delegated to Class Model
Public Property Let Model_FileName(ByVal strNewValue As String)

    mudtModel.FileName = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsFrozen() As Boolean
```

```
    Model_IsFrozen = mudtModel.IsFrozen

End Property

' Delegated to Class Model
Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)

    mudtModel.IsFrozen = blnNewValue

End Property

' Delegated to Class Model
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)

    Call mudtModel.AddChecksum(dblChecksum)

End Sub

' Delegated to Class Model
Public Sub Model_InitChecksums()

    mudtModel.InitChecksums

End Sub

' Delegated to Class Model
Public Sub Model_InitTempChecksums()

    mudtModel.InitTempChecksums

End Sub

' Delegated to Class Model
Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean

    Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)

End Function

' Delegated to Class Model
Public Property Get Model_Comments() As String

    Model_Comments = mudtModel.Comments

End Property
```

```
' Delegated to Class Model
Public Property Let Model_Comments(ByVal strNewValue As String)

   mudtModel.Comments = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_Clones() As CClones

   Set Model_Clones = mudtModel.Clones

End Property

' Delegated to Class Model
Public Property Get Model_Variables() As CVariables

   Set Model_Variables = mudtModel.Variables

End Property

' Delegated to Class Model
Public Property Get Model_Constraints() As CConstraints

   Set Model_Constraints = mudtModel.Constraints

End Property

' Delegated to Class Model
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)

   mudtModel.IsDirty = blnNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsDirty() As Boolean

   Model_IsDirty = mudtModel.IsDirty

End Property

' Delegated to Class Model
Public Property Let Model_LastClone(ByVal intNewValue As Integer)
```

```
    mudtModel.LastClone = intNewValue

End Property

' Delegated to Class Model
Public Property Get Model_LastClone() As Integer

    Model_LastClone = mudtModel.LastClone

End Property

' Delegated to Class Model
Public Sub Model_FreezeModel()

    Call mudtModel.FreezeModel

End Sub

' Delegated to Class Model
Public Sub Model_OpenDoc(ByVal udtWord As MSWord)

    Call mudtModel.OpenDoc(udtWord)

End Sub

' Delegated to Class Model
Public Sub Model_CloseDoc()

    Call mudtModel.CloseDoc

End Sub

' Delegated to Class Model
Public Sub Model_CloseAllCloneDocs()

    Call mudtModel.CloseAllCloneDocs

End Sub

' Delegated to Class Model
Public Sub Model_ReadModel()

    mudtModel.ReadModel

End Sub
```

```
' Delegated to Class Model
Public Sub Model_ReadObjects()

   mudtModel.ReadObjects

End Sub

' Delegated to Class Model
Public Sub Model_WriteModel()

   mudtModel.WriteModel

End Sub

' Delegated to Class Model
Public Sub Model_WriteObjects()

   mudtModel.WriteObjects

End Sub

' Delegated to Class Model
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
   ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
   blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

   Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _
      blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)

End Function

' implemented here
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte)

   Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _
      bytDifference, 50)

End Sub

' Delegated to Class Model
Public Sub Model_SubstituteValues(ByVal objO As Object, _
   ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
   ByVal intStartPos As Integer)
```

```
End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

  With udtClone.CloneDoc.Bookmarks
    If .Exists("tca_RespA") = False Or _
      .Exists("tca_RespB") = False Or _
      .Exists("tca_RespC") = False Or _
      .Exists("tca_RespD") = False Or _
      .Exists("tca_RespE") = False Or _
      .Exists("tca_Key") = False Then

      Call MsgBox("Model is missing a TCA Bookmark!", vbExclamation, "Hey!")
      Exit Sub
    End If
  End With

  Dim nchoices As Integer
  Dim lowerbound As Integer
  Dim upperbound As Integer

  nchoices = 5
  lowerbound = 1
  upperbound = 8

  Dim resp(10) As String
  Dim used(10) As Integer

  resp(0) = udtClone.CloneDoc.Bookmarks("key").Range.Text

  Dim i As Integer
  For i = lowerbound To upperbound
    used(i) = 0
    resp(i) = udtClone.CloneDoc.Bookmarks("resp" & Format(i)).Range.Text
  Next

  Dim nselected As Integer
  nselected = 0

  Dim rnumber As Integer
  Dim rnumbers(10) As Integer

  While (nselected < upperbound)
    rnumber = (upperbound - lowerbound + 1) * Rnd + lowerbound - 0.5
    If (rnumber > upperbound) Then
```

```
      rnumber = upperbound
    End If

    If (used(rnumber) = 0) Then
      used(rnumber) = 1
      nselected = nselected + 1
      rnumbers(nselected) = rnumber
    End If
  Wend

  Dim unsorted(10) As Integer
  unsorted(0) = 0
  nselected = 0

  Dim j As Integer
  Dim n As Integer

  Dim crStr As String
  Dim tabcrStr As String
  crStr = Chr(13)
  tabcrStr = Chr(9) & Chr(13)

  For i = lowerbound To upperbound
    If resp(rnumbers(i)) <> tabcrStr And _
      resp(rnumbers(i)) <> crStr And _
      Mid(resp(rnumbers(i)), 1, 10) <> "Distractor" Then

      n = 0
      For j = 0 To nselected
        If IsNumeric(resp(rnumbers(i))) = True And _
          IsNumeric(resp(unsorted(j))) = True And _
          Asc(resp(rnumbers(i))) <> 36 Then ' 36 is the $ sign
          If Val(resp(rnumbers(i))) = Val(resp(unsorted(j))) Then
             If Asc(resp(rnumbers(i))) <> 1 Then
                n = 1
                Exit For
             End If
          End If
        Else
          If resp(rnumbers(i)) = resp(unsorted(j)) Then
             If Asc(resp(rnumbers(i))) <> 1 Then
                n = 1
                Exit For
             End If
          End If
```

```
            End If
          Next

          If n = 0 Then
            nselected = nselected + 1
            unsorted(nselected) = mumbers(i)
            If nselected = nchoices - 1 Then
            If nselected = upperbound Then
              Exit For
            End If
          End If
        End If
      Next

      For i = 0 To nselected
        used(i) = 0
      Next

      Dim sorted(10) As Integer
      Dim resp1, resp2 As String
      Dim val1, val2 As Variant

      For i = 0 To nselected
        For j = 0 To nselected
          If (used(j) = 0) Then
            sorted(i) = unsorted(j)
           n = j
            Exit For
          End If
        Next

        For j = 0 To nselected
          If (used(j) = 0) Then

            resp1 = resp(unsorted(j))
            resp2 = resp(sorted(i))

            If left(resp1, 1) = "$" Then
              val1 = Val(right(resp1, Len(resp1) - 1))
            Else
              val1 = Val(resp1)
            End If

            If left(resp2, 1) = "$" Then
              val2 = Val(right(resp2, Len(resp2) - 1))
            Else
              val2 = Val(resp2)
            End If

            If (val1 < val2) Then
              sorted(i) = unsorted(j)
              n = j
            End If

          End If
        Next

        used(n) = 1
      Next

      For i = 0 To nselected
        If sorted(i) = 0 Then
          Exit For
        End If
      Next

      Dim min, max As Integer

      min = i - 4
      If min < 0 Then
        min = 0
      End If

      max = i
      If max > nselected - 4 Then
        max = nselected - 4
      End If

      If max < 0 Then
        max = 0
      End If

      Dim iStart As Integer
      Dim iEnd As Integer

      If max > 0 And max + 4 <= nselected Then
        iStart = lastStart
        While iStart = lastStart
          iStart = (max - min + 1) * Rnd + min - 0.5
        Wend
```

```
    lastStart = iStart
    iEnd = iStart + nchoices - 1
  Else
    iStart = 0
    If nselected > 4 Then
      iEnd = 4
    Else
      iEnd = nselected
    End If

    lastStart = iStart
  End If

  Dim respRange As Range
  Dim choice As String
  Dim key As String

  n = 1
  For i = iStart To iEnd
    choice = Mid("ABCDE", n, 1)

    If sorted(i) = 0 Then
      udtClone.CloneDoc.Bookmarks("key").Range.Copy
    Else
      udtClone.CloneDoc.Bookmarks("resp" & Format(sorted(i))).Range.Copy
    End If

    Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & choice).Range
    respRange.Paste
'    respRange.Borders.Enable = False
    respRange.Borders.InsideLineStyle = wdLineStyleNone

    udtClone.CloneDoc.Bookmarks.Add name:="tca_Resp" & choice, Range:=respRange

    respRange.InsertBefore Text:=choice & ".  "

    If sorted(i) = 0 Then
      key = choice
      udtClone.key = choice
    End If

    n = n + 1
  Next

  For i = nselected + 1 To nchoices - 1
```

```
    choice = Mid("ABCDE", i + 1, 1)
    Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & choice).Range
    respRange.Text = "[NO VALUE]" & Chr(13) & Chr(10)
    udtClone.CloneDoc.Bookmarks.Add name:="tca_Resp" & choice, Range:=respRange
    respRange.InsertBefore Text:=choice & ".  "
  Next

  Dim keyRange As Range
  Set keyRange = udtClone.CloneDoc.Bookmarks("tca_Key").Range
  keyRange.InsertBefore Text:="Key is " & key

End Sub
```

→

09594540 . 050400

```
' QCModel.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "QCModel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Model

Dim mudtModel As Model

Private Sub Class_Initialize()

    Set mudtModel = New Model

End Sub

' Delegated to Class Model
Public Property Get Model_FileName() As String

    Model_FileName = mudtModel.FileName

End Property

' Delegated to Class Model
Public Property Let Model_FileName(ByVal strNewValue As String)

    mudtModel.FileName = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsFrozen() As Boolean

    Model_IsFrozen = mudtModel.IsFrozen
```

```
End Property

' Delegated to Class Model
Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)

   mudtModel.IsFrozen = blnNewValue

End Property

' Delegated to Class Model
Public Property Get Model_Comments() As String

   Model_Comments = mudtModel.Comments

End Property

' Delegated to Class Model
Public Property Let Model_Comments(ByVal strNewValue As String)

   mudtModel.Comments = strNewValue

End Property

' Delegated to Class Model
Public Property Get Model_Clones() As CClones

   Set Model_Clones = mudtModel.Clones

End Property

' Delegated to Class Model
Public Property Get Model_Variables() As CVariables

   Set Model_Variables = mudtModel.Variables

End Property

' Delegated to Class Model
Public Property Get Model_Constraints() As CConstraints

   Set Model_Constraints = mudtModel.Constraints

End Property

'Delegated to Class Model
```

```
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)

   Call mudtModel.AddChecksum(dblChecksum)

End Sub

' Delegated to Class Model
Public Sub Model_InitChecksums()

   mudtModel.InitChecksums

End Sub

' Delegated to Class Model
Public Sub Model_InitTempChecksums()

   mudtModel.InitTempChecksums

End Sub

'Delegated to Class Model
Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean

   Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)

End Function

' Delegated to Class Model
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)

   mudtModel.IsDirty = blnNewValue

End Property

' Delegated to Class Model
Public Property Get Model_IsDirty() As Boolean

   Model_IsDirty = mudtModel.IsDirty

End Property

' Delegated to Class Model
Public Property Let Model_LastClone(ByVal intNewValue As Integer)

   mudtModel.LastClone = intNewValue
```

```
End Property

' Delegated to Class Model
Public Sub Model_FreezeModel()

    Call mudtModel.FreezeModel

End Sub

' Delegated to Class Model
Public Property Get Model_LastClone() As Integer

    Model_LastClone = mudtModel.LastClone

End Property

' Delegated to Class Model
Public Sub Model_OpenDoc(ByVal udtWord As MSWord)

    Call mudtModel.OpenDoc(udtWord)

End Sub

' Delegated to Class Model
Public Sub Model_CloseDoc()

    Call mudtModel.CloseDoc

End Sub

' Delegated to Class Model
Public Sub Model_CloseAllCloneDocs()

    Call mudtModel.CloseAllCloneDocs

End Sub

'Delegated to Class Model
Public Sub Model_ReadModel()

    mudtModel.ReadModel

End Sub

' Delegated to Class Model
```

```
Public Sub Model_ReadObjects()

   mudtModel.ReadObjects

End Sub

' Delegated to Class Model
Public Sub Model_WriteModel()

   mudtModel.WriteModel

End Sub

' Delegated to Class Model
Public Sub Model_WriteObjects()

   mudtModel.WriteObjects

End Sub

' Delegated to Class Model
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
   ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
   blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

   Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _
      blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)

End Function

' implemented here
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte)

   Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _
      bytDifference, 275)

End Sub

' Delegated to Class Model
Public Sub Model_SubstituteValues(ByVal objO As Object, _
   ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
   ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
   ByVal intStartPos As Integer)
```

```
End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

  Dim rnumber As Integer
  Dim sLen As Integer
  Dim columnRange As Range
  Dim columnAValStr As String
  Dim columnBValStr As String

  With udtClone.CloneDoc

    rnumber = .Tables(2).Rows.Count * Rnd + 0.5
    .Tables(2).Cell(Row:=rnumber, Column:=1).Range.Copy
    columnAValStr = .Tables(2).Cell(Row:=rnumber, Column:=2).Range.Text

    sLen = Len(columnAValStr)
    If sLen > 1 Then
      columnAValStr = left(columnAValStr, sLen - 1)
    End If

    Set columnRange = .Bookmarks("tca_ColumnA").Range
    columnRange.Paste

    rnumber = .Tables(3).Rows.Count * Rnd + 0.5
    .Tables(3).Cell(Row:=rnumber, Column:=1).Range.Copy
    columnBValStr = .Tables(3).Cell(Row:=rnumber, Column:=2).Range.Text

    sLen = Len(columnBValStr)
    If sLen > 1 Then
      columnBValStr = left(columnBValStr, sLen - 1)
    End If

    Set columnRange = .Bookmarks("tca_ColumnB").Range
    columnRange.Paste

    If .Tables(1).Columns.Count = 4 Then ' fixes weird behavior if only 1 row in model
      .Tables(1).Cell(Row:=1, Column:=4).Delete
      .Tables(1).Cell(Row:=1, Column:=3).Delete
    End If

    Dim key As String
    Dim columnAValue
    Dim columnBValue
```

```
        If IsNumeric(columnAValStr) = True And _
          IsNumeric(columnBValStr) = True Then

          columnAValue = Val(columnAValStr)
          columnBValue = Val(columnBValStr)

          If columnAValue > columnBValue Then
            key = "A"
          ElseIf columnBValue > columnAValue Then
            key = "B"
          ElseIf columnAValue = columnBValue Then
            key = "C"
          End If

        End If

    End With

    Dim keyRange As Range

    Set keyRange = udtClone.CloneDoc.Bookmarks("tca_Key").Range
    If key = "" Then
        keyRange.InsertBefore Text:="TCA cannot determine the key"
    Else
        keyRange.InsertBefore Text:="Key is " & key
    End If

    udtClone.key = key

End Sub
```

```
' StringSolver.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = 0   'False
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "StringSolver"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Dim mudtVS As VarString

Dim mcolValues As Collection

Public Property Let StringVariable(ByVal udtNewValue As VarString)

   Set mudtVS = udtNewValue

End Property

Public Property Get RandomValueCollection() As Collection

   Dim udtSS As SubString
   Dim strS As String
   Dim varS As Variant

   Set mcolValues = New Collection

   strS = mudtVS.StringCollection.Item(GetRandomIndex)

   If mudtVS.IsIndexed Then
      Set udtSS = New SubString
      udtSS.Delimiter = mudtVS.Delimiter
      udtSS.StringValue = strS
      For Each varS In udtSS.StringCollection
         Call mcolValues.Add(varS)
      Next varS
   Else
```

```
      Call mcolValues.Add(strS)
    End If

    Set RandomValueCollection = mcolValues

End Property

Private Function GetRandomIndex() As Integer

    Dim intI As Integer

    intI = mudtVS.StringCollection.Count * Rnd + 0.5

    ' Seems to produce an out-of-range value sometimes.
    ' This will fix it.
    If intI < 1 Then intI = 1
    If intI > mudtVS.StringCollection.Count Then intI = mudtVS.StringCollection.Count

    GetRandomIndex = intI

End Function
```

```
' StringSolverx.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "StringSolver"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mcolSV As Collection

Private Sub Class_Initialize()

    Set mcolSV = New Collection

End Sub
```

```
' SubString.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "SubString"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private mstrDelimiter As String

Private mstrString As String

Private mcolStr As Collection

Private Sub Class_Initialize()

   Set mcolStr = New Collection

End Sub

Public Property Let Delimiter(ByVal strNewValue As String)

   mstrDelimiter = strNewValue

End Property

' use this to convert a concatenated string to a collection
Public Property Let StringValue(ByVal strNewValue As String)

   mstrString = strNewValue

End Property

' or use this to convert a collection to a concatenated string
Public Property Let StringCollection(ByVal colNewValue As Collection)

   Set mcolStr = colNewValue

End Property
```

```
' converts collection into concatenated string
Public Property Get StringValue() As String

   Dim varS As Variant
   Dim strS As String

   ' build new string
   For Each varS In mcolStr
      strS = strS & varS & mstrDelimiter
   Next varS

   ' trim last character
   If Len(strS) > 0 Then
      StringValue = left(strS, Len(strS) - 1)
   End If

End Property

' converts concatenated string into a collection
Public Property Get StringCollection() As Collection

   Dim colC As New Collection
   Dim intI As Integer

   For intI = 1 To NumSubStrings
      Call colC.Add(GetSubString(intI))
   Next intI

   Set StringCollection = colC

End Property

' returns the number of substrings in this string
Public Property Get NumSubStrings() As Integer

   Dim intD As Integer
   Dim intI As Integer
   Dim varS As Variant

   If Len(mstrString) = 0 Then
      NumSubStrings = 0
      Exit Property
   End If

   For intI = 1 To Len(mstrString)
```

```
        If Mid(mstrString, intI, 1) = mstrDelimiter Then
            intD = intD + 1
        End If
    Next intI

    NumSubStrings = intD + 1

End Property

Public Sub AddSubString(ByVal strNewValue As String)

    Call mcolStr.Add(strNewValue)

End Sub

' parses the substring from the string depending on intIndex
Public Function GetSubString(ByVal intIndex As Integer) As String

    ' see if index is valid for the current string
    If NumSubStrings < intIndex Then
        GetSubString = ""
        Exit Function
    End If

    ' index into the string using delimiter
    Dim varI1 As Variant
    Dim varI2 As Variant
    Dim intCount As Integer

    varI2 = 0

    Do
        varI1 = varI2
        varI2 = InStr(varI1 + 1, mstrString, mstrDelimiter)
        intCount = intCount + 1
        If varI2 = 0 Then
            varI2 = Len(mstrString) + 1
        End If

    Loop Until intCount = intIndex

    GetSubString = Mid(mstrString, varI1 + 1, varI2 - varI1 - 1)

End Function
```

```
' Value.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "Value"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Dim mstrVariableName As String
Dim mstrValue As String
Dim mblnChecksum As Boolean
Dim mstrPrologString As String
Dim mudtVariableType As VariableType

Public Property Get VariableName() As String

   VariableName = mstrVariableName

End Property

Public Property Let VariableName(ByVal strNewValue As String)

   mstrVariableName = strNewValue

End Property

Public Property Get Value() As String

   Value = mstrValue

End Property

Public Property Let Value(ByVal strNewValue As String)

   mstrValue = strNewValue

End Property

Public Property Get Checksum() As Boolean
```

```
    Checksum = mblnChecksum
End Property
Public Property Let Checksum(ByVal blnNewValue As Boolean)
    mblnChecksum = blnNewValue
End Property
Public Property Get PrologString() As String
    PrologString = mstrPrologString
End Property
Public Property Let PrologString(ByVal strNewValue As String)
    mstrPrologString = strNewValue
End Property
Public Property Get VariableType() As VariableType
    VariableType = mudtVariableType
End Property
Public Property Let VariableType(ByVal udtNewValue As VariableType)
    mudtVariableType = udtNewValue
End Property
```

```
' VarFraction.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "VarFraction"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Variable

Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private mstrFromNum As String
Private mstrFromDen As String
Private mstrToNum As String
Private mstrToDen As String
Private mstrByNum As String
Private mstrByDen As String
Private mblnMixedNumbers As Boolean
Private mblnIsIndependent As Boolean

Private Sub Class_Initialize()

    Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

    Set mudtVar = Nothing

End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

    Variable_Name = mudtVar.Name
```

```
End Property

' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

   mudtVar.Name = RHS

End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

   mudtVar.Typ = udtNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

   Variable_Typ = mudtVar.Typ

End Property

' Delegated to Class Variable
Public Property Get Variable_Index() As Long

   Variable_Index = mudtVar.Index

End Property

' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

   mudtVar.Index = lngNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

   Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
```

```
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

    mudtVar.Enabled = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

    Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

    mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean

    Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

    mudtVar.Checksum = blnNewValue

End Property

Public Property Get FromNumerator() As String

    FromNumerator = mstrFromNum

End Property

Public Property Let FromNumerator(ByVal strNewValue As String)

    mstrFromNum = strNewValue
    mudtVar.IsDirty = True
```

```
End Property

Public Property Get FromDenominator() As String

   FromDenominator = mstrFromDen

End Property

Public Property Let FromDenominator(ByVal strNewValue As String)

   mstrFromDen = strNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get ToNumerator() As String

   ToNumerator = mstrToNum

End Property

Public Property Let ToNumerator(ByVal strNewValue As String)

   mstrToNum = strNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get ToDenominator() As String

   ToDenominator = mstrToDen

End Property

Public Property Let ToDenominator(ByVal strNewValue As String)

   mstrToDen = strNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get ByNumerator() As String

   ByNumerator = mstrByNum
```

```
End Property

Public Property Let ByNumerator(ByVal strNewValue As String)

   mstrByNum = strNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get ByDenominator() As String

   ByDenominator = mstrByDen

End Property

Public Property Let ByDenominator(ByVal strNewValue As String)

   mstrByDen = strNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get MixedNumbers() As Boolean

   MixedNumbers = mblnMixedNumbers

End Property

Public Property Let MixedNumbers(ByVal blnNewValue As Boolean)

   mblnMixedNumbers = blnNewValue
   mudtVar.IsDirty = True

End Property

Public Property Get IsIndependent() As Boolean

   IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

   mblnIsIndependent = blnNewValue
   mudtVar.IsDirty = True
```

```
End Property

Public Sub Update(ByVal strName As String, _
   ByVal strFromN As String, ByVal strFromD As String, _
   ByVal strToN As String, ByVal strToD As String, _
   ByVal strByN As String, ByVal strByD As String, _
   ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
   ByVal blnMixedNumber As Boolean)

   Variable_Name = strName
   FromNumerator = strFromN
   FromDenominator = strFromD
   ToNumerator = strToN
   ToDenominator = strToD
   ByNumerator = strByN
   ByDenominator = strByD
   IsIndependent = blnIsIndependent
   Variable_Checksum = blnChecksum
   MixedNumbers = blnMixedNumber

End Sub


Public Function Variable_PrologFormat() As String

   Dim str1 As String

   If mblnIsIndependent Then
      str1 = "fraction(" & mudtVar.Name & "),offgrid(" & _
         mudtVar.Name & "),[" & _
         mstrFromNum & "/" & mstrFromDen & "<=" & _
         mudtVar.Name & "<=" & mstrToNum & "/" & _
         mstrToDen & " step " & mstrByNum & "/" & mstrByDen & "]"
   Else
      str1 = "fraction(" & mudtVar.Name & ")"
   End If

   Variable_PrologFormat = str1

End Function

Public Function Variable_ScreenFormat() As String

   Dim str1 As String
   Dim strOpt As String
```

```
    If mudtVar.Checksum Then
      strOpt = "(C,"
    Else
      strOpt = "(c,"
    End If

    If mblnMixedNumbers Then
      strOpt = strOpt & "M),"
    Else
      strOpt = strOpt & "m),"
    End If

    If mblnIsIndependent Then
      str1 = mudtVar.Name & strOpt & ": Fraction, " & _
        mstrFromNum & "/" & mstrFromDen & " to " & _
        mstrToNum & "/" & mstrToDen & " by " & _
        mstrByNum & "/" & mstrByDen
    Else
      str1 = mudtVar.Name & strOpt & ": Fraction"
    End If

    Variable_ScreenFormat = str1

End Function

Public Property Get Variable_ReadType(udtFile As File) As VariableType

    Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

    Dim vField As Variant

    Call udtFile.ReadField(vField) ' reads version stamp
    Call udtFile.ReadField(vField)
    mudtVar.Name = vField

    Call udtFile.ReadField(vField)
    mudtVar.Enabled = vField

    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField
    Call udtFile.ReadField(vField)
    IsIndependent = vField

    Call udtFile.ReadField(vField)
    FromNumerator = vField

    Call udtFile.ReadField(vField)
    FromDenominator = vField

    Call udtFile.ReadField(vField)
    ToNumerator = vField

    Call udtFile.ReadField(vField)
    ToDenominator = vField

    Call udtFile.ReadField(vField)
    ByNumerator = vField

    Call udtFile.ReadField(vField)
    ByDenominator = vField

    Call udtFile.ReadField(vField)
    MixedNumbers = vField

End Sub

Public Sub Variable_WriteObjectData(udtFile As File)

    Dim udtType As VariableType

    udtType = vtFraction
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
    Call udtFile.WriteField(IsIndependent)
    Call udtFile.WriteField(FromNumerator)
    Call udtFile.WriteField(FromDenominator)
    Call udtFile.WriteField(ToNumerator)
    Call udtFile.WriteField(ToDenominator)
    Call udtFile.WriteField(ByNumerator)
    Call udtFile.WriteField(ByDenominator)
    Call udtFile.WriteField(MixedNumbers)
    mudtVar.IsDirty = False

End Sub

' makes a copy of this object
Public Function Variable_Copy() As Variable

    Dim udtVF As New VarFraction
    Dim udtV As Variable

    Set udtV = udtVF

    udtV.Name = mudtVar.Name
    udtV.Enabled = mudtVar.Index
    udtV.IsDirty = mudtVar.IsDirty
    udtV.Checksum = mudtVar.Checksum

    udtVF.FromNumerator = FromNumerator
    udtVF.FromDenominator = FromDenominator
    udtVF.ByNumerator = ByNumerator
    udtVF.ByDenominator = ByDenominator
    udtVF.ToNumerator = ToNumerator
    udtVF.ToDenominator = ToDenominator
    udtVF.IsIndependent = IsIndependent
    udtVF.MixedNumbers = MixedNumbers

    Set Variable_Copy = udtV

End Function
```

```
' Variable.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = 0   'False
 Persistable = 0  'NotPersistable
 DataBindingBehavior = 0  'vbNone
 DataSourceBehavior  = 0  'vbNone
 MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Variable"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

Private mstrName As String
Private mudtType As VariableType
Private mlngIndex As Long
Private mblnEnabled As Boolean
Private mblnIsDirty As Boolean
Private mblnChecksum As Boolean

Public Enum VariableType
   vtInteger = 0
   vtReal = 1
   vtFraction = 2
   vtString = 3
   vtUntyped = 4
End Enum

Public Property Get name() As String

   name = mstrName

End Property

Public Property Let name(ByVal strNewValue As String)

   If mstrName <> strNewValue Then
      mstrName = strNewValue
      mblnIsDirty = True
```

```
    End If

End Property

Public Property Get Typ() As VariableType

    Typ = mudtType

End Property

Public Property Let Typ(ByVal udtNewValue As VariableType)

    If mudtType <> udtNewValue Then
        mudtType = udtNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get index() As Long

    index = mlngIndex

End Property

Public Property Let index(ByVal lngNewValue As Long)

    If mlngIndex <> lngNewValue Then
        mlngIndex = lngNewValue
        mblnIsDirty = True
    End If

End Property

Public Property Get Enabled() As Boolean

    Enabled = mblnEnabled

End Property

Public Property Let Enabled(ByVal blnNewValue As Boolean)

    If mblnEnabled <> blnNewValue Then
        mblnEnabled = blnNewValue
        mblnIsDirty = True
    End If
End Property
Public Property Let IsDirty(ByVal blnNewValue As Boolean)
    mblnIsDirty = blnNewValue
End Property
Public Property Get IsDirty() As Boolean
    IsDirty = mblnIsDirty
End Property
Public Property Let Checksum(ByVal blnNewValue As Boolean)
    If mblnChecksum <> blnNewValue Then
        mblnChecksum = blnNewValue
        mblnIsDirty = True
    End If
End Property
Public Property Get Checksum() As Boolean
    Checksum = mblnChecksum
End Property
' implemented in the subclasses of Variable
Public Function PrologFormat() As String
End Function
' implemented in the subclasses of Variable
Public Function ScreenFormat() As String
End Function
' implemented in the subclasses of Variable
```

```
Public Sub ReadObjectData(udtFile As File)
End Sub
' implemented in the subclasses of Variable
Public Sub WriteObjectData(udtFile As File)
End Sub
Public Property Get ReadType(udtFile As File) As VariableType
    Dim udtType As VariableType
    Call udtFile.ReadField(udtType)
    ReadType = udtType
End Property
' implemented in the subclasses of Variable
Public Function Copy() As Variable
End Function
```

```
' VarInteger.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "VarInteger"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Variable

Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private mstrFrom As String
Private mstrTo As String
Private mstrBy As String
Private mblnIsIndependent As Boolean

Private Sub Class_Initialize()

   Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

   Set mudtVar = Nothing

End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

   Variable_Name = mudtVar.Name

End Property

' Delegated to Class Variable
```

```
Public Property Let Variable_Name(ByVal RHS As String)

   mudtVar.Name = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

   Variable_Typ = mudtVar.Typ

End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

   mudtVar.Typ = udtNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Index() As Long

   Variable_Index = mudtVar.Index

End Property

' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

   mudtVar.Index = lngNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

   Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

   mudtVar.Enabled = RHS
```

```
End Property

' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

   Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

   mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean

   Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

   mudtVar.Checksum = blnNewValue

End Property

Public Property Get From() As String

   From = mstrFrom

End Property

Public Property Let From(ByVal strNewValue As String)

   If mstrFrom <> strNewValue Then
      mstrFrom = strNewValue
      mudtVar.IsDirty = True
   End If

End Property
```

```
Public Property Get Too() As String

   Too = mstrTo

End Property

Public Property Let Too(ByVal strNewValue As String)

   If mstrTo <> strNewValue Then
      mstrTo = strNewValue
      mudtVar.IsDirty = True
   End If

End Property

Public Property Get By() As String

   By = mstrBy

End Property

Public Property Let By(ByVal strNewValue As String)

   If mstrBy <> strNewValue Then
      mstrBy = strNewValue
      mudtVar.IsDirty = True
   End If

End Property

Public Property Get IsIndependent() As Boolean

   IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

   If mblnIsIndependent <> blnNewValue Then
      mblnIsIndependent = blnNewValue
      mudtVar.IsDirty = True
   End If

End Property
```

```
Public Sub Update(ByVal strName As String, _
   ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
   ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean)

   Variable_Name = strName
   From = strFrom
   Too = strTo
   By = strBy
   IsIndependent = blnIsIndependent
   Variable_Checksum = blnChecksum

End Sub

Public Function Variable_PrologFormat() As String

   Dim str1 As String

   If mblnIsIndependent Then
      str1 = "int(" & mudtVar.Name & "),[" & mstrFrom & "<=" & _
            mudtVar.Name & "<=" & mstrTo & " step " & mstrBy & "]"
   Else
      str1 = "int(" & mudtVar.Name & ")"
   End If

   Variable_PrologFormat = str1

End Function

Public Function Variable_ScreenFormat() As String

   Dim str1 As String
   Dim strT As String
   Dim strOpt As String

   If mudtVar.Checksum Then
      strOpt = "(C)"
   Else
      strOpt = "(c)"
   End If

   If mblnIsIndependent Then
      str1 = mudtVar.Name & strOpt & ": Int, " & mstrFrom & " to " & _
         mstrTo & " by " & mstrBy
   Else
      str1 = mudtVar.Name & strOpt & ": Int"
```

```
    End If
    Variable_ScreenFormat = str1
End Function
Public Property Get Variable_ReadType(udtFile As File) As VariableType
    Variable_ReadType = mudtVar.ReadType(udtFile)
End Property
Public Sub Variable_ReadObjectData(udtFile As File)
    Dim vField As Variant
    Call udtFile.ReadField(vField) ' reads version stamp
    Call udtFile.ReadField(vField)
    mudtVar.Name = vField
    Call udtFile.ReadField(vField)
    mudtVar.Enabled = vField
    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField
    Call udtFile.ReadField(vField)
    From = vField
    Call udtFile.ReadField(vField)
    Too = vField
    Call udtFile.ReadField(vField)
    By = vField
    Call udtFile.ReadField(vField)
    IsIndependent = vField
End Sub
Public Sub Variable_WriteObjectData(udtFile As File)
    Dim udtType As VariableType
    udtType = vtInteger
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
    Call udtFile.WriteField(From)
    Call udtFile.WriteField(Too)
    Call udtFile.WriteField(By)
    Call udtFile.WriteField(IsIndependent)

    mudtVar.IsDirty = False

End Sub

' makes a copy of this object
Public Function Variable_Copy() As Variable

    Dim udtVI As New VarInteger
    Dim udtV As Variable

    Set udtV = udtVI

    udtV.Name = mudtVar.Name
    udtV.Typ = vtInteger
    udtV.Enabled = mudtVar.Index
    udtV.IsDirty = mudtVar.IsDirty
    udtV.Checksum = mudtVar.Checksum

    udtVI.From = From
    udtVI.Too = Too
    udtVI.By = By
    udtVI.IsIndependent = IsIndependent

    Set Variable_Copy = udtV

End Function
```

```
' VarReal.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "VarReal"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Variable

Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 2

Private mstrFrom As String
Private mstrTo As String
Private mstrBy As String
Private mblnTrailingZeros As Boolean
Private mstrPrecision As String
Private mblnIsIndependent As Boolean
Private mblnIsOnGrid As Boolean

Private Sub Class_Initialize()

    Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

    Set mudtVar = Nothing

End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

    Variable_Name = mudtVar.Name
```

```
End Property

' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

   mudtVar.Name = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

   Variable_Typ = mudtVar.Typ

End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

   mudtVar.Typ = udtNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

   Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

   mudtVar.Enabled = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Index() As Long

   Variable_Index = mudtVar.Index

End Property

' Delegated to Class Variable
```

```
Public Property Let Variable_Index(ByVal lngNewValue As Long)

   mudtVar.Index = lngNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

   Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

   mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean

   Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

   mudtVar.Checksum = blnNewValue

End Property

Public Property Get From() As String

   From = mstrFrom

End Property

Public Property Let From(ByVal strNewValue As String)

   If mstrFrom <> strNewValue Then
      mstrFrom = strNewValue
      mudtVar.IsDirty = True
```

```
    End If

End Property

Public Property Get Too() As String

    Too = mstrTo

End Property

Public Property Let Too(ByVal strNewValue As String)

    If mstrTo <> strNewValue Then
        mstrTo = strNewValue
        mudtVar.IsDirty = True
    End If

End Property

Public Property Get By() As String

    By = mstrBy

End Property

Public Property Let By(ByVal strNewValue As String)

    If mstrBy <> strNewValue Then
        mstrBy = strNewValue
        mudtVar.IsDirty = True
    End If

End Property

Public Property Get TrailingZeros() As Boolean

    TrailingZeros = mblnTrailingZeros

End Property

Public Property Let TrailingZeros(ByVal blnNewValue As Boolean)

    If mblnTrailingZeros <> blnNewValue Then
        mblnTrailingZeros = blnNewValue
        mudtVar.IsDirty = True
    End If

End Property

Public Property Get IsOnGrid() As Boolean

    IsOnGrid = mblnIsOnGrid

End Property

Public Property Let IsOnGrid(ByVal blnNewValue As Boolean)

    If mblnIsOnGrid <> blnNewValue Then
        mblnIsOnGrid = blnNewValue
        mudtVar.IsDirty = True
    End If

End Property

Public Property Get Precision() As String

    Precision = mstrPrecision

End Property

Public Property Let Precision(ByVal strNewValue As String)

    If mstrPrecision <> strNewValue Then
        mstrPrecision = strNewValue
        mudtVar.IsDirty = True
    End If

End Property

Public Property Get DecimalPlaces() As Integer

    If InStr(1, mstrPrecision, ".") = 0 Then
        DecimalPlaces = 0
    Else
        DecimalPlaces = Len(mstrPrecision) - 1
    End If

End Property

Public Property Get IsIndependent() As Boolean
```

```
   IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

   If mblnIsIndependent <> blnNewValue Then
      mblnIsIndependent = blnNewValue
      mudtVar.IsDirty = True
   End If

End Property

Public Sub Update(ByVal strName As String, _
   ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
   ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
   ByVal blnTrailingZeros As Boolean, _
   ByVal strPrecision As String, ByVal blnIsOnGrid As Boolean)

   Variable_Name = strName
   From = strFrom
   Too = strTo
   By = strBy
   IsIndependent = blnIsIndependent
   Variable_Checksum = blnChecksum
   TrailingZeros = blnTrailingZeros
   Precision = strPrecision
   IsOnGrid = blnIsOnGrid

End Sub

Public Function Variable_PrologFormat() As String

   Dim str1 As String

   If mblnIsIndependent Then
      str1 = "real({" & mudtVar.Name & "," & mstrPrecision & "}),[" _
         & mstrFrom & "<=" & mudtVar.Name & "<=" & mstrTo & " step " & _
         mstrBy & "]"
   Else
      str1 = "real(" & mudtVar.Name & ")"
   End If

   If Not mblnIsOnGrid Then
      str1 = str1 & ",offgrid(" & mudtVar.Name & ")"
```

```
    End If

    Variable_PrologFormat = str1

End Function

Public Function Variable_ScreenFormat() As String

    Dim str1 As String
    Dim strOpt As String

    If mudtVar.Checksum Then
        strOpt = "(C,"
    Else
        strOpt = "(c,"
    End If

    If mblnTrailingZeros Then
        strOpt = strOpt & "T,"
    Else
        strOpt = strOpt & "t,"
    End If

    If mblnIsOnGrid Then
        strOpt = strOpt & "G,"
    Else
        strOpt = strOpt & "g,"
    End If

    strOpt = strOpt & mstrPrecision & ")"

    If mblnIsIndependent Then
        str1 = mudtVar.Name & strOpt & ": Real, " & mstrFrom & " to " & _
            mstrTo & "  by " & mstrBy
    Else
        str1 = mudtVar.Name & strOpt & ": Real"
    End If

    Variable_ScreenFormat = str1

End Function

Public Property Get Variable_ReadType(udtFile As File) As VariableType

    Variable_ReadType = mudtVar.ReadType(udtFile)
```

```
End Property

Public Sub Variable_ReadObjectData(udtFile As File)

    Dim vField As Variant
    Dim intVersion As Integer

    Call udtFile.ReadField(vField) ' reads version stamp
    intVersion = vField

    Call udtFile.ReadField(vField)
    mudtVar.Name = vField

    Call udtFile.ReadField(vField)
    mudtVar.Enabled = vField

    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField

    Call udtFile.ReadField(vField)
    From = vField

    Call udtFile.ReadField(vField)
    Too = vField

    Call udtFile.ReadField(vField)
    By = vField

    Call udtFile.ReadField(vField)
    TrailingZeros = vField

    Call udtFile.ReadField(vField)
    Precision = vField

    Call udtFile.ReadField(vField)
    IsIndependent = vField

    If intVersion < 2 Then ' this field is new to version 2 of VarReal
        IsOnGrid = True
    Else
        Call udtFile.ReadField(vField)
        IsOnGrid = vField
    End If

End Sub
```

```
Public Sub Variable_WriteObjectData(udtFile As File)

   Dim udtType As VariableType

   udtType = vtReal
   Call udtFile.WriteField(udtType)
   Call udtFile.WriteField(mintVERSIONSTAMP)
   Call udtFile.WriteField(mudtVar.Name)
   Call udtFile.WriteField(mudtVar.Enabled)
   Call udtFile.WriteField(mudtVar.Checksum)
   Call udtFile.WriteField(From)
   Call udtFile.WriteField(Too)
   Call udtFile.WriteField(By)
   Call udtFile.WriteField(TrailingZeros)
   Call udtFile.WriteField(Precision)
   Call udtFile.WriteField(IsIndependent)
   Call udtFile.WriteField(IsOnGrid)

   mudtVar.IsDirty = False

End Sub

' makes a copy of this object
Public Function Variable_Copy() As Variable

   Dim udtVR As New VarReal
   Dim udtV As Variable

   Set udtV = udtVR

   udtV.Name = mudtVar.Name
   udtV.Typ = vtReal
   udtV.Enabled = mudtVar.Index
   udtV.IsDirty = mudtVar.IsDirty
   udtV.Checksum = mudtVar.Checksum

   udtVR.From = From
   udtVR.Too = Too
   udtVR.By = By
   udtVR.Precision = Precision
   udtVR.TrailingZeros = TrailingZeros
   udtVR.IsIndependent = IsIndependent
   udtVR.IsOnGrid = IsOnGrid

   Set Variable_Copy = udtV
```

```
End Function
```

```
' VarString.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "VarString"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Variable

Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private mstrDelimiter As String

Private mblnIsIndexed As Boolean

Private mcolString As New Collection

Private Sub Class_Initialize()

    Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

    Set mudtVar = Nothing

End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

    Variable_Name = mudtVar.Name

End Property
```

```
' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

   mudtVar.Name = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

   Variable_Typ = mudtVar.Typ

End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

   mudtVar.Typ = udtNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Index() As Long

   Variable_Index = mudtVar.Index

End Property

' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

   mudtVar.Index = lngNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

   Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)
```

```
   mudtVar.Enabled = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

   Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

   mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean

   Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

   mudtVar.Checksum = blnNewValue

End Property

Public Property Get Delimiter() As String

   Delimiter = mstrDelimiter

End Property

Public Property Let Delimiter(ByVal strNewValue As String)

   If mstrDelimiter <> strNewValue Then
      mstrDelimiter = strNewValue
      mudtVar.IsDirty = True
   End If
End Property

Public Property Get IsIndexed() As Boolean

   IsIndexed = mblnIsIndexed

End Property

Public Property Let IsIndexed(ByVal blnNewValue As Boolean)

   mblnIsIndexed = blnNewValue

End Property

Public Property Get StringCollection() As Collection

   Set StringCollection = mcolString

End Property

Public Property Let StringCollection(ByVal colNewValue As Collection)

   Dim intIndex As Integer

   If mcolString.Count <> colNewValue.Count Then
      Set mcolString = colNewValue
      mudtVar.IsDirty = True
      Exit Property
   End If

   For intIndex = 1 To mcolString.Count
      If mcolString.Item(intIndex) <> colNewValue.Item(intIndex) Then
         Set mcolString = colNewValue
         mudtVar.IsDirty = True
         Exit Property
      End If
   Next intIndex

End Property

' returns the largest number of delimited substrings in the string collection
Public Property Get NumIndices() As Integer

   Dim intD As Integer
   Dim intHiD As Integer
```

```
    Dim intI As Integer
    Dim varS As Variant
    Dim udtSubStr As New SubString

    ' if there are no strings in the collection
    If mcolString.Count = 0 Then
        NumIndices = 1
        Exit Property
    End If

    udtSubStr.Delimiter = mstrDelimiter

    For Each varS In mcolString
        udtSubStr.StringValue = varS
        intD = udtSubStr.NumSubStrings
        If intD > intHiD Then
            intHiD = intD
        End If
    Next varS

    NumIndices = intHiD

End Property

Public Function Variable_PrologFormat() As String

    Variable_PrologFormat = ""

End Function

Public Function Variable_ScreenFormat() As String

    Dim str1 As String
    Dim strS As String
    Dim intIndex As Integer
    Dim strOpt As String

    If mudtVar.Checksum Then
        strOpt = "(C,"
    Else
        strOpt = "(c,"
    End If

    strOpt = strOpt & Str(NumIndices) & "," & mstrDelimiter & ")"
    For intIndex = 1 To 3

        If mcolString.Count >= intIndex Then
            strS = strS & mcolString.Item(intIndex)
            If mcolString.Count > intIndex Then
                strS = strS & ","
            End If
        End If

    Next intIndex

    If mcolString.Count > 3 Then
        strS = strS & "..."
    End If

    str1 = mudtVar.Name & strOpt & ": String, in [" & strS & "]"

    Variable_ScreenFormat = str1

End Function

Public Property Get Variable_ReadType(udtFile As File) As VariableType

    Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

    Dim vField As Variant
    Dim intCount As Integer

    Call udtFile.ReadField(vField) ' reads version stamp
    Call udtFile.ReadField(vField)
    mudtVar.Name = vField

    Call udtFile.ReadField(vField)
    mudtVar.Enabled = vField

    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField

    Call udtFile.ReadField(vField)
    mstrDelimiter = vField
    Call udtFile.ReadField(vField)
    mblnIsIndexed = vField

    Call udtFile.ReadField(vField)
    intCount = vField

    Dim intI As Integer

    ' read in the strings
    For intI = 1 To intCount

        Call udtFile.ReadField(vField)
        Call mcolString.Add(vField)

    Next intI

End Sub

Public Sub Variable_WriteObjectData(udtFile As File)

    Dim udtType As VariableType

    udtType = vtString
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
    Call udtFile.WriteField(mstrDelimiter)
    Call udtFile.WriteField(mblnIsIndexed)

    Dim intCount As Integer

    intCount = mcolString.Count
    Call udtFile.WriteField(intCount)

    Dim intI As Integer

    ' write out the strings
    For intI = 1 To mcolString.Count
        Call udtFile.WriteField(mcolString.Item(intI))
    Next intI

    mudtVar.IsDirty = False
```

```
End Sub

' makes a copy of this object
Public Function Variable_Copy() As Variable

  Dim udtVS As New VarString
  Dim udtV As Variable
  Dim varS As Variant

  Set udtV = udtVS

  udtV.Name = mudtVar.Name
  udtV.Typ = vtString
  udtV.Enabled = mudtVar.Index
  udtV.IsDirty = mudtVar.IsDirty
  udtV.Checksum = mudtVar.Checksum

  udtVS.Delimiter = Delimiter
  udtVS.IsIndexed = IsIndexed

  Set Variable_Copy = udtV

  For Each varS In mcolString
    Call udtVS.StringCollection.Add(varS)
  Next varS

End Function




' VarUntyped.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "VarUntyped"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Implements Variable
```

```
Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private Sub Class_Initialize()

   Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

   Set mudtVar = Nothing

End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

   Variable_Name = mudtVar.Name

End Property

' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

   mudtVar.Name = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

   Variable_Typ = mudtVar.Typ

End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

   mudtVar.Typ = udtNewValue

End Property
```

```
' Delegated to Class Variable
Public Property Get Variable_Index() As Long

   Variable_Index = mudtVar.Index

End Property

' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

   mudtVar.Index = lngNewValue

End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

   Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

   mudtVar.Enabled = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

   Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

   mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean
```

```
   Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

   mudtVar.Checksum = blnNewValue

End Property

Public Function Variable_PrologFormat() As String

   Variable_PrologFormat = ""

End Function

Public Function Variable_ScreenFormat() As String

   Dim str1 As String
   Dim strS As String
   Dim intIndex As Integer
   Dim strOpt As String

   If mudtVar.Checksum Then
      strOpt = "(C)"
   Else
      strOpt = "(c)"
   End If

   str1 = mudtVar.Name & strOpt & ": Untyped"

   Variable_ScreenFormat = str1

End Function

Public Property Get Variable_ReadType(udtFile As File) As VariableType

   Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

   Dim vField As Variant
```

```
    Dim intCount As Integer

    Call udtFile.ReadField(vField) ' reads version stamp
    Call udtFile.ReadField(vField)
    mudtVar.Name = vField

    Call udtFile.ReadField(vField)
    mudtVar.Enabled = vField

    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField

End Sub

Public Sub Variable_WriteObjectData(udtFile As File)

    Dim udtType As VariableType

    udtType = vtUntyped
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)

    mudtVar.IsDirty = False

End Sub

' makes a copy of this object
Public Function Variable_Copy() As Variable

    Dim udtV As New Variable

    udtV.Name = mudtVar.Name
    udtV.Typ = vtUntyped
    udtV.Enabled = mudtVar.Index
    udtV.IsDirty = mudtVar.IsDirty
    udtV.Checksum = mudtVar.Checksum

    Set Variable_Copy = udtV

End Function
```

```
' Win32API.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "Win32API"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
' used for making calls to the Win32 API
Option Explicit

Private Type FILETIME
     dwLowDateTime As Long
     dwHighDateTime As Long
End Type

Private Const MAX_PATH = 260

Private Type WIN32_FIND_DATA
     dwFileAttributes As Long
     ftCreationTime As FILETIME
     ftLastAccessTime As FILETIME
     ftLastWriteTime As FILETIME
     nFileSizeHigh As Long
     nFileSizeLow As Long
     dwReserved0 As Long
     dwReserved1 As Long
     cFileName As String * MAX_PATH
     cAlternate As String * 14
End Type

Private Const INVALID_HANDLE_VALUE = -1

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
  (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" _
  (ByVal hFileName As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long

Private Declare Function GetCurrentDirectory Lib "kernel32" _
```

```
    Alias "GetCurrentDirectoryA" (ByVal nBufferLength As Long, _
    ByVal lpBuffer As String) As Long

Private Declare Function SendMessageLong Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Long, _
    ByVal Msg As Long, _
    ByVal wParam As Long, _
    ByVal lParam As Long) As Long

Private Declare Function SystemParametersInfo Lib "user32" _
    Alias "SystemParametersInfoA" (ByVal uAction As Long, _
    ByVal uParam As Long, ByRef lpvParam As Any, _
    ByVal fuWinIni As Long) As Long

Private Const SPI_GETDRAGFULLWINDOWS = 38
Private Const SPI_SETDRAGFULLWINDOWS = 37
Private Const SPIF_SENDWININICHANGE = 2

Public Function IsFullWindowDragOn() As Boolean

    Dim result As Long

    'Call API and check for successful call.
    If SystemParametersInfo(SPI_GETDRAGFULLWINDOWS, 0&, result, 0&) <> 0 Then
        'Feature supported now check value of result.
        If result = 0 Then
            IsFullWindowDragOn = False
        Else
            IsFullWindowDragOn = True
        End If
        'Call failed, feature not supported.
    Else
        IsFullWindowDragOn = False
    End If

End Function

Public Sub TurnOffFullWindowDrag()

    Dim result As Long

    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 0&, _
        ByVal vbNullString, SPIF_SENDWININICHANGE)

End Sub
```

```
Public Sub TurnOnFullWindowDrag()

  Dim result As Long

  result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 1&, _
    ByVal vbNullString, SPIF_SENDWININICHANGE)

End Sub

' returns true if strFN exists
Public Function FileExists(ByVal strFN) As Boolean

  Dim lngHandle As Long
  Dim w32FindData As WIN32_FIND_DATA

  lngHandle = FindFirstFile(strFN, w32FindData)

  If lngHandle = INVALID_HANDLE_VALUE Then
    FileExists = False
  Else
    FileExists = True
    Call FindClose(lngHandle)
  End If

End Function

' returns a collection of file names that satisfy strMask.  The path seems to
' disappear from the returned file names.

Public Function FindAllFiles(ByVal strMask As String) As Collection

  Dim lngHandle As Long
  Dim lngRet As Long
  Dim w32FindData As WIN32_FIND_DATA
  Dim strFN As String
  Dim varI As Variant
  Dim colFNs As New Collection

  lngHandle = FindFirstFile(strMask, w32FindData)

  If lngHandle = INVALID_HANDLE_VALUE Then
    Exit Function
  End If

  Do
```

```
        strFN = TrimAtFirstNull(w32FindData.cFileName)
        Call colFNs.Add(strFN) ' add to the collection

    Loop Until FindNextFile(lngHandle, w32FindData) = 0

    Set FindAllFiles = colFNs

End Function

' returns the current directory
Public Function CurrentDirectory() As String

    Dim strBuf As String
    Dim lngRet As Long
    Dim varI As Variant

    strBuf = Space(300)
    lngRet = GetCurrentDirectory(300, strBuf)
    CurrentDirectory = TrimAtFirstNull(strBuf)

End Function

' enable full row select in list view control
Public Sub EnableListViewFullRowSelect(lvwLV As ListView)

    Dim lngStyle As Long
    Dim lngL As Long

    'get the current ListView style
    lngStyle = SendMessageLong(lvwLV.hwnd, LVM_GETEXTENDEDLISTVIEWSTYLE, 0&,
0&)

    'set the extended style bit
    lngStyle = lngStyle Or LVS_EX_FULLROWSELECT

    'set the new ListView style
    lngL = SendMessageLong(lvwLV.hwnd, LVM_SETEXTENDEDLISTVIEWSTYLE, 0&,
lngStyle)

End Sub
```

```
' Word.cls
VERSION 1.0 CLASS
BEGIN
 MultiUse = -1  'True
END
Attribute VB_Name = "MSWord"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Private Const WM_CLOSE = &H10
Private mWDApp As Word.Application

Private Type RECT
  left As Long
  top As Long
  right As Long
  bottom As Long
End Type

Private Declare Function GetParent Lib "user32" _
  (ByVal hWndChild As Long) As Long

Private Declare Function SetParent Lib "user32" _
  (ByVal hWndChild As Long, ByVal hWndNewParent As Long) As Long

Private Declare Function FindWindow Lib "user32" _
  Alias "FindWindowA" (ByVal lpClassName As String, _
  ByVal lpWindowName As String) As Long

Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Any) As Long

Private Declare Function GetWindowRect Lib "user32" _
  (ByVal hwnd As Long, lpRect As RECT) As Long

Private Declare Function SetWindowPos Lib "user32" _
  (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
  ByVal X As Long, ByVal Y As Long, ByVal cx As Long, _
  ByVal cy As Long, ByVal wFlags As Long) As Long
```

```
Dim mlngHandle As Long
Dim origParent As Long
Dim origLeft As Long
Dim origTop As Long
Dim origWidth As Long
Dim origHeight As Long

Private Sub Class_Initialize()

'  mlngHandle = FindWindow("OpusApp", vbNullString)

'  Do While mlngHandle <> 0
'     SendMessage mlngHandle, WM_CLOSE, mlngHandle, 0
'     mlngHandle = FindWindow("OpusApp", vbNullString)
'  Loop

   mlngHandle = FindWindow("OpusApp", vbNullString)

   If mlngHandle <> 0 Then
      Set mWDApp = GetObject(, "Word.Application.8")
   Else
      On Error Resume Next
      Set mWDApp = GetObject(, "Word.Application.8")

      If err.Number = 0 Then
         MsgBox "Phantom WinWord detected!"
         Call mWDApp.Quit(False)
      Else
         err.Clear
      End If

      Set mWDApp = CreateObject("Word.Application.8")
   End If

   mlngHandle = FindWindow("OpusApp", vbNullString)

   If mlngHandle <> 0 Then
      origParent = GetParent(mlngHandle)

      If mWDApp.left < 0 Then
        origLeft = 0
      Else
        origLeft = mWDApp.left
      End If
```

```
        If mWDApp.top < 0 Then
          origTop = 0
        Else
          origTop = mWDApp.top
        End If

        origWidth = mWDApp.Width
        origHeight = mWDApp.Height

        Call SetParent(mlngHandle, frmTCA.fraWord.hwnd)
      End If

      mWDApp.Visible = True

End Sub

Private Sub Class_Terminate()

      mWDApp.Visible = False

      Call SetParent(mlngHandle, origParent)
      Call mWDApp.Move(origLeft, origTop)
      Call mWDApp.Resize(origWidth, origHeight)

      Call mWDApp.Quit(False) ' don't save!

End Sub

Public Property Get WordApp() As Word.Application

      Set WordApp = mWDApp

End Property

Public Property Get DocumentsCount() As Long

      DocumentsCount = mWDApp.Documents.Count

End Property

Public Property Get SelectionType() As Long

      SelectionType = mWDApp.Selection.Type

End Property
```

```
Public Property Get SelectionText() As String

   SelectionText = mWDApp.Selection.Text

End Property

Public Sub Resize()

   Dim WindowRect As RECT

   GetWindowRect frmTCA.fraWord.hwnd, WindowRect

   Dim lngH As Long
   Dim lngW As Long

   lngW = frmTCA.ScaleX(WindowRect.right - WindowRect.left, vbPixels, vbPoints)
   lngH = frmTCA.ScaleY(WindowRect.bottom - WindowRect.top, vbPixels, vbPoints)

   Call mWDApp.Resize(lngW, lngH)
   Call mWDApp.Move(0, 0)

'  SetWindowPos mlngHandle, 0, 0, 0, _
'     WindowRect.right - WindowRect.left, _
'     WindowRect.bottom - WindowRect.top, 64

   CommandBars("File").Controls("Exit").Enabled = False

End Sub

Public Sub CloseAllDocs()

   Dim docD As Document

   For Each docD In mWDApp.Documents

      If Not docD.ReadOnly Then
         docD.Close
      Else
         Call docD.Close(False)
      End If
   Next docD

End Sub
```